Zarak Khan
CSC 362

Assignment 4

Purpose: To analyze the game of Nim using the minimax and alpha-beta pruning algorithms, and how optimization techniques and heuristic functions can be utilized to efficiently determine optimal moves in game trees.

Problem 1

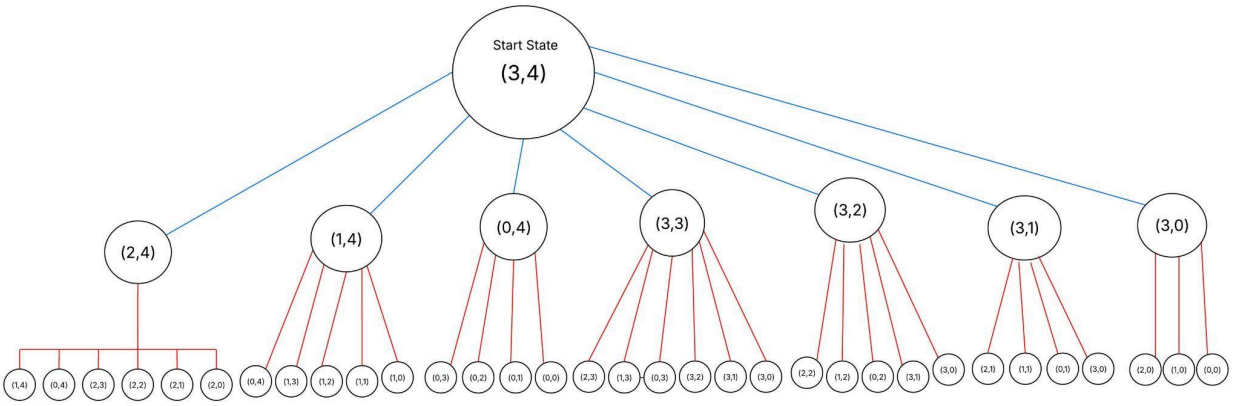
Game Description

The game of Nim is a two-player game that involves a number of distinct heaps of objects. For this analysis, we start the game with two heaps (heap X and heap Y). Heap X contains 3 objects, and heap Y contains 4 objects.

Two players, player 1 and player 2, take turns removing at least one object from either of the heaps. They may remove any number of objects from a heap, as long as at least one object is removed.

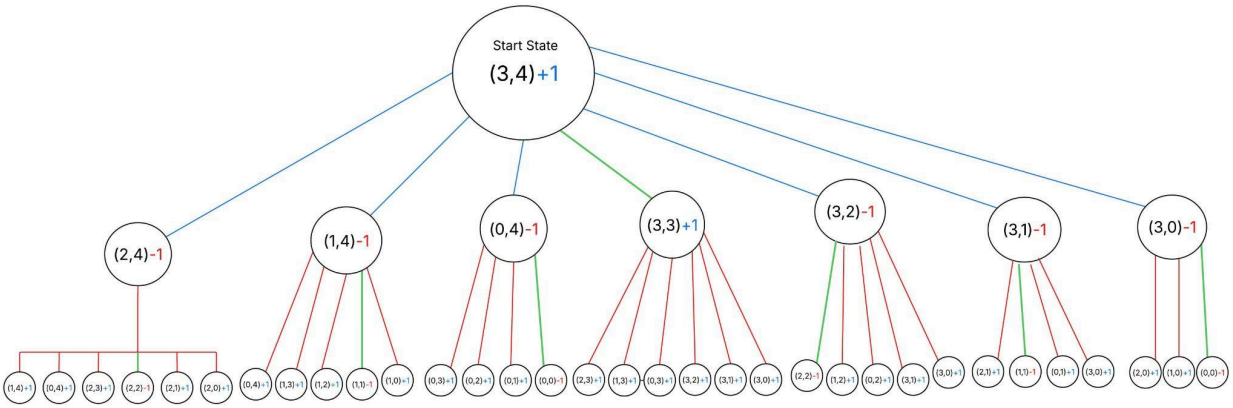
The game continues until all heaps are empty. The player who removes the last object wins the game.

Game Tree



This game tree shows all possible moves for a 2-ply game of Nim (one full turn for each player). Each branch represents a player's move, and each node represents the current heap configuration (x,y). The blue lines represent all the possible moves player 1 can make, and the red lines represent all the possible moves player 2 can make. The start state/root node (3,4) represents the starting position, where player 1 makes the first move.

Minimax Algorithm



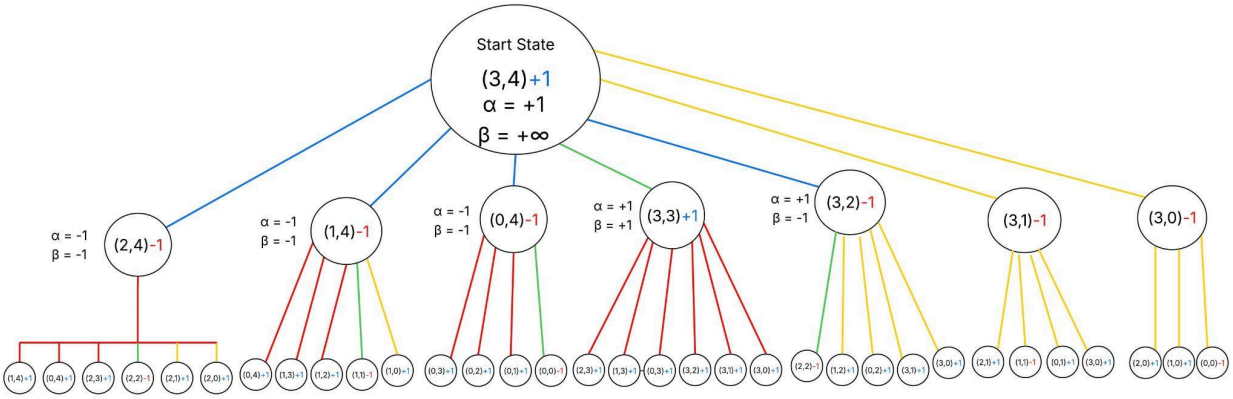
The minimax algorithm and nim-sum has been applied to the two-ply game tree. The first level represents all possible moves that can be made by player 1 (max), and the second level represents all possible moves that can be made by player 2 (min). +1 indicates a winning position for player 1, while -1 indicates a losing position. When it is

player 1's turn, they will choose the highest value among their children (highest utility value). When it is player 2's turn, they will choose the lowest value among their children (lowest utility value).

The blue lines represent all the possible moves player 1 can make, and the red lines represent all the possible moves player 2 can make. The green lines represent the optimal move that can be made by each player.

The optimal move for either player is to keep the heaps equally sized, so that the heaps have the same number of objects. In this case, the optimal move for player 1 is to remove one object from heap Y (3,3), putting player 2 in a losing position (no optimal moves). The same strategy can be used for player 2.

Alpha-Beta Pruning:



The alpha-beta pruning algorithm has been applied to the game tree to reduce the number of nodes that need to be evaluated. The yellow lines represent the nodes that are pruned. The root node starts with $\alpha = -\infty$ and $\beta = +\infty$. α is the highest value a max node can guarantee, while the β value is the lowest value a min node can guarantee. The pruning algorithm eliminates the need to search a total of 16 nodes. For example, after node (3,3) returns an α value of +1, it does not need to keep searching because the algorithm already found the best possible value for the branch. The remaining nodes are pruned because they can't affect the outcome.

Analysis:

The minimax algorithm can solve the game of Nim by evaluating all the possible moves, and determining the most optimal move for each player. One player tries to maximize their chances of winning, while the other tries to minimize the other player's chances of winning. In the game with a heap configuration of (3,4), the size of the game tree is somewhat manageable. As the number of heaps or heap size increases, the complexity and size of the game tree grows exponentially, making the algorithm less effective due to the exhaustive search that is required.

Alpha-beta pruning is even more effective than the minimax algorithm because it reduces the number of nodes that need to be searched. This eliminates the search of nodes which don't affect the outcome, and the same optimal outcome is found while reducing the number of nodes searched. In the current (3,4) heap configuration, 16 nodes are pruned, making the search more effective compared to the minimax algorithm. If the number of heaps or heap size were to be increased, the number of pruned nodes would grow substantially. Therefore, pruning significantly increases the effectiveness of the search in large, complex game trees.

Problem 2

Optimization Techniques:

The minimax algorithm can be further optimized using iterative deepening, which is when the algorithm incrementally increases the depth limit until the maximum depth is reached. Iterative deepening helps make the search more manageable and efficient in large game trees by gradually increasing search depth, rather than evaluating all the possible moves at once. This approach enables the algorithm to find the most optimal moves in shallow depths, making it much more effective for larger game trees. This can be combined with alpha-beta pruning to increase the effectiveness of the search.

In the current two-ply game, a depth 1 search would only evaluate the 7 nodes (player 1's moves) without looking at its children. In the depth 2 search, the algorithm evaluates player 2's moves, and the children can be reordered to prioritize the most promising nodes.

Heuristic Evaluation:

A heuristic evaluation function can estimate the value of game states in Nim. A heuristic function that can be used is the nim-sum function, which is a bitwise XOR of both heap

sizes. A nim-sum of 0 means the current player is in a losing position, and a non-zero means they are in a winning position.

Using a 2-ply game with heap configuration (3,4), the minimax algorithm can use this heuristic to assign a value to each node, such as +1 for a winning position and -1 for a losing position. The minimax algorithm can use these heuristic values to determine the optimal move for each player.

A standard minimax algorithm can be used on smaller game trees to search until it reaches a terminal state (end of the game), and assigns an exact utility value (+1 or -1) to these terminal nodes to determine the optimal moves. However, a heuristic-based minimax limits the search to a certain depth, and then uses a heuristic evaluation function (nim-sum) to estimate the value of the non-terminal nodes. Although the heuristic-based minimax may not be as accurate as the standard minimax (it's only an estimation), it is much more efficient for larger and more complex game trees. It is impractical to use a standard minimax algorithm for larger trees because it is infeasible to evaluate every possible move.