ECSE-526 Assignment # 1

Muhammad Amir Hamza
McGill ID : 261210478

September 24, 2024

1. Abstract

In this assignment, the author developed and analyzed game-playing agents for Power Connect-4, utilizing both naive and advanced heuristic functions in various versions of the Minimax algorithm. Power Connect-4 is a modified version of Connect-4 that incorporates shifting moves, allowing players to displace the opponent's pawn, even pushing it out of the grid. The first task involved simulating the game rules, followed by the development of depth-limited Minimax, $\alpha\beta$-pruning, and $\alpha\beta$-pruning with move-ordering in Part I. The utility of non-terminal nodes was determined using a heuristic function provided by the instructor. In Part II, the author created an advanced heuristic function for non-terminal states to improve the robustness of the agent. The strategies were then evaluated and compared based on the number of states visited and the total time taken by the agent, reflecting computational complexity.

2. Introduction

Connect-4 is a two-player game played on an 8x8 grid, where each player tries to win by aligning four consecutive pieces in a row—diagonally, vertically, or horizontally. Power Connect-4 is a modified version of Connect-4, allowing players to slide an opponent's piece horizontally by one space, either left or right, if an adjacent space is empty or if the piece is located at the edge of the grid. If the opponent's piece is at the edge, sliding it results in its removal from the board. Any sliding action creates a gap, which must then be filled by dropping pieces from above in the same column.

Modeling the game rules was a prerequisite for developing the game-playing agent. To accomplish this, the author created a GameBoard class in Python. This class tracks the game state, monitors terminal conditions, provides available moves for a given state, handles gap filling, calculates each player's score when a terminal state is reached, and displays the grid for visual clarity.

In Part I, the game-playing agent was developed using the Depth-Limited Minimax algorithm. Due to the exponential growth of states in the search tree, a simple Minimax algorithm with full tree search would require an excessive amount of time to evaluate each move. To address this, the time complexity of Depth-Limited Minimax was reduced using $\alpha\beta$-pruning. While $\alpha\beta$-pruning theoretically reduces the number of states visited, its efficiency largely depends on the order in which moves are evaluated. To enhance this, the author employed a move-ordering technique to prioritize actions based on their expected utility.

3. Part-I : Game Playing Agent using given Initial Heuristic Function

In first section of the assignment, author developed the following agent:

- Depth-limited Minimax Algorithm.
- Depth-limited Minimax with $\alpha\beta$-pruning.
- Depth-limited Minimax with $\alpha\beta$-pruning and move-ordering

Each algorithm was simulated for the first three moves, and the time taken as well as the number of states evaluated were recorded. For all of the aforementioned agents, the utility of non-terminal states was evaluated using the initial heuristic function, defined as:

$$\begin{aligned} H(n) =&(\text{number of runs of 2 or more white pieces}) \\ &- (\text{number of runs of 2 or more black pieces}) \end{aligned} \tag{1}$$

Further discussion regarding the the above agents are as follow:

States Visited by Initial Moves:

As Minimax will explore all the nodes of the tree to a specified depth or till the terminal condition is reached. The total number of nodes explored will be of order $O(b^d)$, where b is the branching factor and d is the depth of the tree. Whereas, the $\alpha\beta$-pruning will theoretically cut the the number of nodes to $O(b^{\lceil d/2 \rceil})$ but it highly depends upon the order in which we explore the moves. The states visited by first three moves made by Minimax and $\alpha\beta$-pruning agent is shown in Figure 1.

Time Taken by Initial Moves:

The time taken by the agent is influenced by the calculations involved in the heuristic function and the number of nodes processed. While the pattern will resemble that of the number of states visited, the ratio of states examined by both agents and their respective computation times may not remain constant. Figure 2 illustrates the time taken by both the Minimax agent and the $\alpha\beta$-pruned agent.

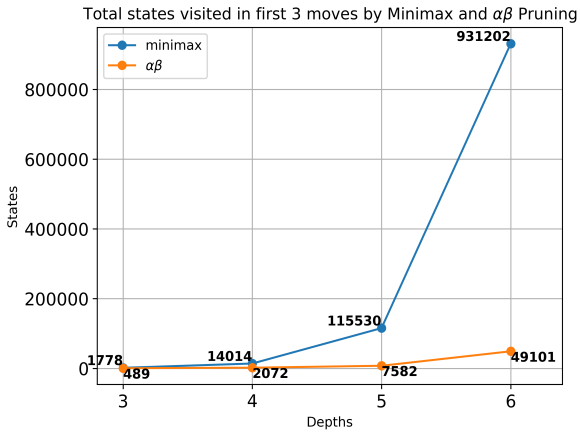

Figure 1: Total states visited Minimax and $\alpha\beta$-pruning agent in first three moves for different depths limits.

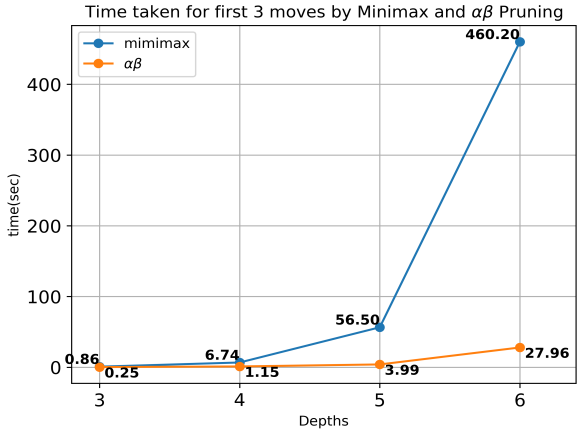

Figure 2: Average time taken by Minimax and $\alpha\beta$-pruning agent for each three moves for different depths limits.

Move orderinga and $\alpha\beta$-pruning:

The $\alpha\beta$-pruning is highly dependent on the order in which nodes are explored. For instance, if the best moves are considered first, alpha-beta pruning can effectively prune most other branches, leading to optimal performance. Conversely, in the worst-case scenario, if the best action is evaluated last among all available options, the number of nodes explored by alpha-beta pruning will approach that of the standard Minimax algorithm. There are different approaches to enhance the effectiveness of $\alpha\beta$-pruning:

- Move Ordering: This approach involves reordering nodes based on their utility to prioritize the most promising moves.
- Transposition Table: This technique stores historically evaluated nodes, allowing the algorithm to reuse previously computed values when encountering the same position again.
- Iterative Deepening: This method gradually increases the search depth and reorders moves based on earlier evaluations.

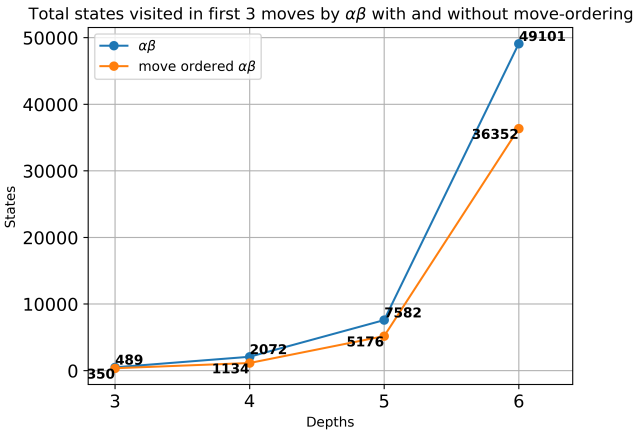

Figure 3: Total states visited by normal $\alpha\beta$-pruning and move-ordered $\alpha\beta$-pruning in first three moves for different depths limits.

The author employed the move-ordering technique to rearrange the nodes for exploration. For this rearrangement, the author used the initial heuristic function provided by the instructor to assess the utility of non-terminal nodes. While the author acknowledges that this approach is suboptimal, it serves its purpose for demonstration.

The result in terms of total number of states explored for $\alpha\beta$ and $\alpha\beta$ with Move-ordering is shown in the Figure 3.

4. Part-II : Game Playing Agent using Modified(Advance) Heuristic Function

In this section, the author developed a modified version of heuristic function for the game playing agent.

Rational Choice of Improved Heuristic Function

The the developed heuristic function is actually a linear combination of multiple sub-heuristic functions. The overall heuristic function can be written as:

$$\begin{aligned} h_{advance}(n) = & h_{con.-3-with-blankspaces}(n) \\ & + h_{con.-3-without-blankspaces}(n) \\ & + h_{central-control}(n) + h_{sliding-elimination}(n) \end{aligned} \tag{2}$$

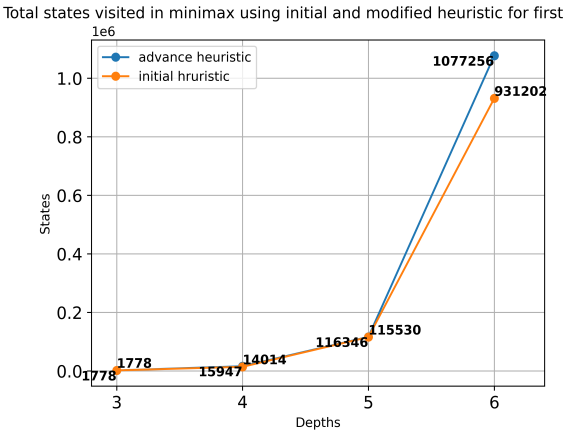

Figure 4: Total number of states explored in first 3 moves by Minimax agent using initial given heuristic and modified advance heuristic. Note: this is only for 3 moves, the gap between the total states will increase for more moves and depths.

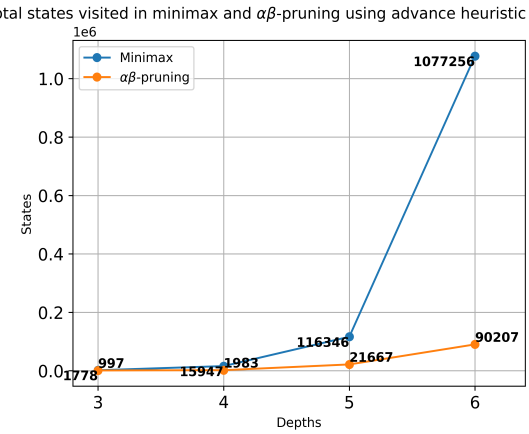

Figure 5: Total number of states explored in first 3 moves by Minimax agent and Minimax with $\alpha\beta$-pruned agent using advance heuristic. The total number of states are reduced drastically when $\alpha\beta$-pruning is used

The explanation and reason behind each of these heuristic functions are as follow:

- $h_{con.-3-with-blankspaces}(n)$: This heuristic counts the number of available sequences of three consecutive pieces in a row (vertically, horizontally, or diagonally) that have at least one adjacent empty space. Its importance lies in the fact that an empty space increases the agent's chances of completing a four-in-a-row. The chances improve significantly if there are empty spaces on both sides of the sequence. As a result, this heuristic has the greatest impact on the overall evaluation, both rewarding and penalizing the function accordingly.

- $h_{con.-3-without-blankspaces}(n)$: This heuristic calculates the number of sequences of three consecutive pieces in a row (vertically, horizontally, or diagonally) without any adjacent empty spaces. The impact on the overall evaluation is smaller compared to $h_{con.-3-with-blankspaces}(n)$, resulting in a lower penalty or reward.

- $h_{central-control}(n)$: This heuristic is based on the idea that pieces located in the center of the board provide greater control and influence. With this motivation, $h_{central-control}(n)$ calculates the number of the player's pieces positioned in the central area of the grid. The central region is defined by the following kernel:

$$kernel = \begin{bmatrix} 1 & 2 & 4 & 5 & 5 & 4 & 2 & 1 \\ 1 & 2 & 4 & 7 & 7 & 4 & 2 & 1 \\ 1 & 2 & 7 & 9 & 9 & 7 & 2 & 1 \\ 1 & 7 & 9 & 9 & 9 & 9 & 7 & 1 \\ 1 & 7 & 9 & 9 & 9 & 9 & 7 & 1 \\ 1 & 2 & 7 & 9 & 9 & 7 & 2 & 1 \\ 1 & 2 & 4 & 7 & 7 & 4 & 2 & 1 \\ 1 & 2 & 4 & 5 & 5 & 4 & 2 & 1 \end{bmatrix}$$

- $h_{sliding-elimination}(n)$: This heuristic evaluates the effect of sliding actions that remove an opponent's piece from the grid. By discarding an opponent's piece, the player can secure a more advantageous position, which improves their chances of controlling the board.

Effects of Heuristic Function on the Number of Nodes Explored:

The author believes that the heuristic function significantly impacts the number of nodes visited. A heuristic with a more exploratory nature will naturally increase the number of available actions at each layer. For instance, in our case, the heuristic encourages moving pieces toward the center of the grid, which allows for greater flexibility in dropping and, in particular, sliding pieces. This increased freedom leads to more potential moves being considered during the search process.

In Figure 4, it can be seen that for even three moves, advance heuristic (with more exploratory tendencies) explored more states as compared to initial heuristic. The gap between the explored states will get wider for more search depth and more moves.

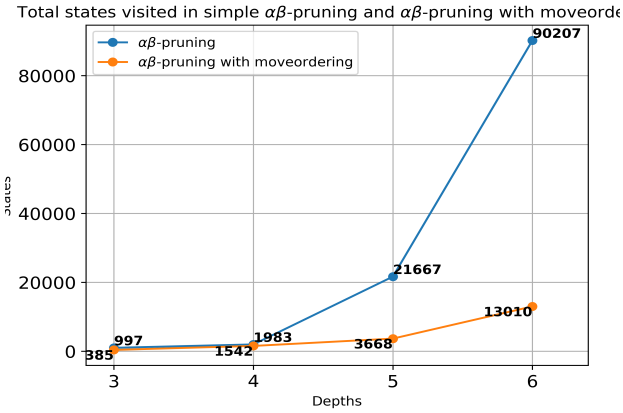

Figure 6: Total number of states explored in first 3 moves by simple $\alpha\beta$-prunned agent and move ordered $\alpha\beta$-pruned agent using advance heuristic. Mover ordering is proven to be effective in reducing the number of states explored by $\alpha\beta$-pruning.

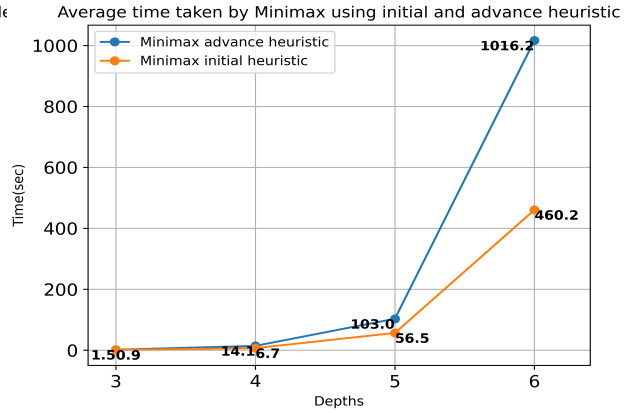

Figure 7: Average time taken by Minimax using advance Heuristic and Minimax using initial heuristic function. One can see that the advance heuristic requires more computational time as compared to simple one.

Effect of $\alpha\beta$ pruning on Number of Nodes Explored:

In Minimax algorithm, all possible states are explored. The number of total nodes explored can be reduced substentially by employing $\alpha\beta$ pruning. In Figure 5, we compared the efficiency of Minimax algorithm with $\alpha\beta$0-pruning and simple Minimax. One can see that $\alpha\beta$0-pruning has considerably reduced the total number of states explored.

Effect of Move ordering on the Number of Nodes Explored in $\alpha\beta$ pruning:

As discussed in the subsequent section, the efficiency of $\alpha\beta$ pruning highly depends upon the order in which it explores the actions or child nodes. If the actions are arranged in ascending order of their utility, then this will be the worst case in which the algorithm will perform similar to Minimax. The author used the move ordering of actions before feeding it to the $\alpha\beta$ algorithm. The results can be seen in Figure 6.

From the above discussion and graphs, it will be correct to state that $\alpha\beta$ pruning with move ordering have the highest effect on the number of nodes to visited.

Computational Complexity of Advance Heuristic:

As the complexity of the heuristic increases, it undoubtedly translates into more mathematical operations, leading to longer computation times compared to the initial, simpler heuristic. Figure 7 illustrates the time taken by the Minimax algorithm

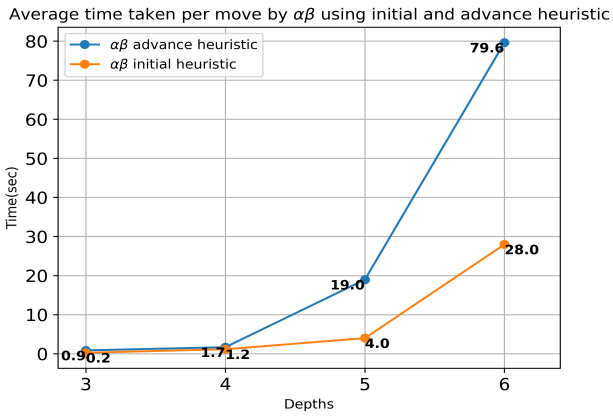

Figure 8: Time taken by $\alpha\beta$-pruned agents using simple and advance heuristic function.

when using both the initial heuristic and our advanced heuristic. The results clearly show that the advanced heuristic requires more time and computational resources to complete its evaluations. Additionally, Figure 8 presents the computational times associated with α-β pruning. From these results, we can conclude that to limit the move evaluations to 10 seconds, a maximum depth of 4 can be effectively achieved.

Memory requirements are not a significant concern in the family of Minimax algorithms, as the complexity is $O(bd)$, where b is the branching factor and d is the depth of the algorithm. Since the branching factor is only marginally influenced by the heuristic function, the primary focus tends to be on time complexity.

However, the increased time complexity is justified by the improved robustness of the advanced heuristic. It demonstrates a greater ability to

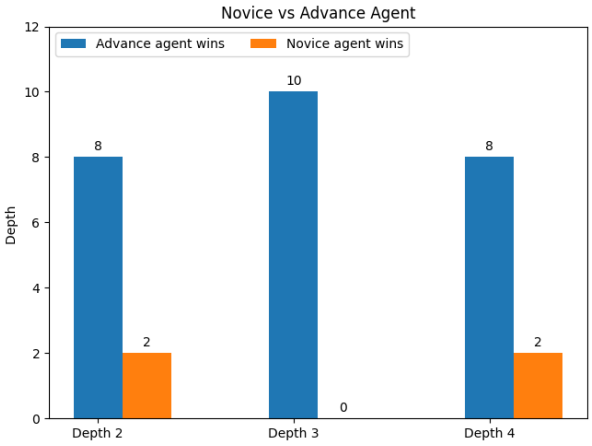

Figure 9: Wins at different depth limits by novice player using the given heuristic and advanced player with updated heuristic function. It can be observed that although Advance heuristic takes more computation power but performs better than initial heuristic function.

win the majority of games compared to the initial heuristic function, making it a more effective choice in competitive scenarios. From the Figure 9 it shows that the agent was able to win most of the times against the agent with initial heuristic function.

Possibility of Guaranteed Winning Sequence for Complex Games

In two-player competitive games like checkers and chess, the existence of a guaranteed winning sequence depends on the game's complexity and the available strategies. Connect-4 is simpler than both checkers and chess, as it originally has only eight drop moves, which limits its complexity.

For checkers, while it is less complex than chess, it still presents significant challenges. Numerous undefeated programs exist that play optimally, meaning that if both players employ perfect strategies, the game will always end in a draw. This indicates that while a winning sequence may exist against a less skilled opponent, there is no guaranteed win against an equally matched player.

Chess represents an even more complex scenario. Programs today play at superhuman levels, consistently defeating human grandmasters. However, despite this advanced capability, no guaranteed winning sequence has been found due to the vast number of possible game states.

Given our current computational abilities, it seems unlikely that a guaranteed winning sequence can be established for these complex games. While one might argue that infinite computational power could eventually lead to such a breakthrough, this remains a distant possibility and is not expected to occur in the foreseeable future.

Log File:

A log file of the match between the agent using the initial heuristic function and the agent using the advanced heuristic function is provided along with the files

5. Conclusion

In this report, the first author analyzes the performance of the initial heuristic provided by the instructor for the Power Connect-4 game. The algorithms considered include Minimax, Minimax with simple $\alpha\beta$-pruning, and move-ordered $\alpha\beta$-pruning. In the second part, the author develops and evaluates the performance of an advanced heuristic function. The analysis focuses on total states visited and the time taken for initial moves, highlighting the differences between the two approaches. Although the advanced heuristic requires more computational resources, it demonstrates improved performance compared to the initial heuristic function.