# Assignment 1: Power-Connect 4

Berton Benjamin

benjamin.berton@mail.mcgill.ca

## Table of Contents

# Part I

We will explore the impact of alpha-beta pruning on the performance of the minimax algorithm in a customized Power Connect-4 game. Specifically, we analyze the time it takes the algorithm to perform state space expansion and the number of nodes visited to evaluate the game tree at different depth cutoffs. We compare the results for the plain minimax algorithm and minimax with alpha-beta pruning to highlight the performance improvements.

To measure performance, two versions of the minimax algorithm have been implemented:

- **Plain Minimax**: This version does not employ any pruning and exhaustively searches the entire game tree to a specified depth.
- **Minimax with Alpha-Beta Pruning**: This version prunes parts of the search tree that do not need to be explored, effectively reducing the number of nodes visited.

For both versions, the time taken to perform the state space expansion and the number of node visited has been measured for depth level (1-5) averaged over the first 5 moves of the game.

# Results

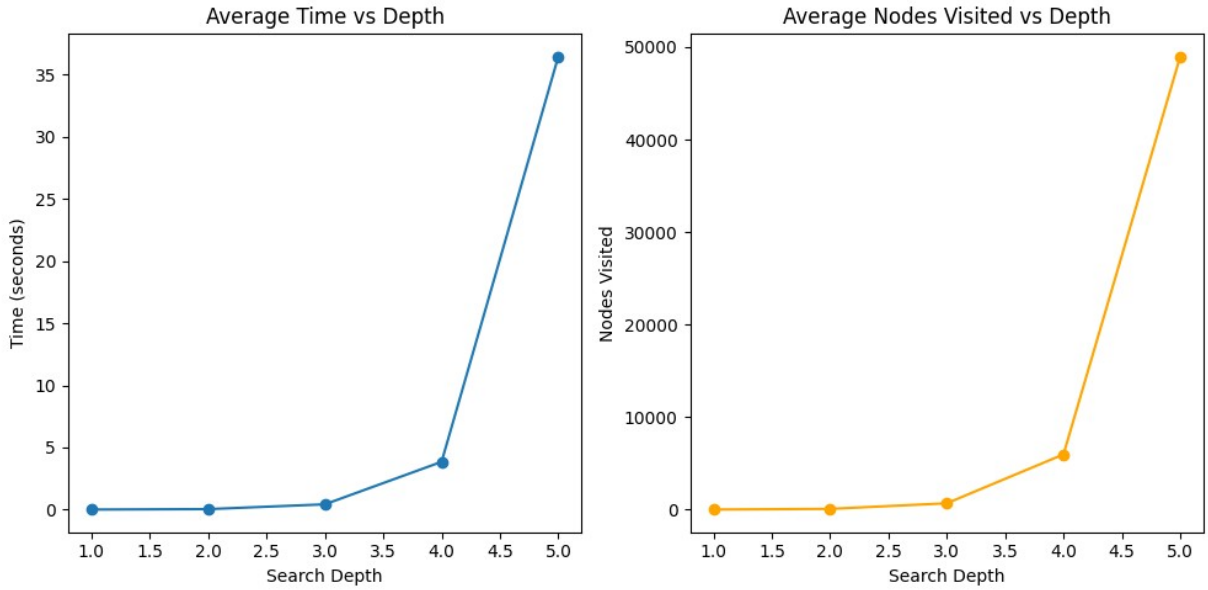


*Figure 1: The graph above illustrates the average time taken and nodes visited by the plain minimax algorithm for different depth cutoffs (1-5).*

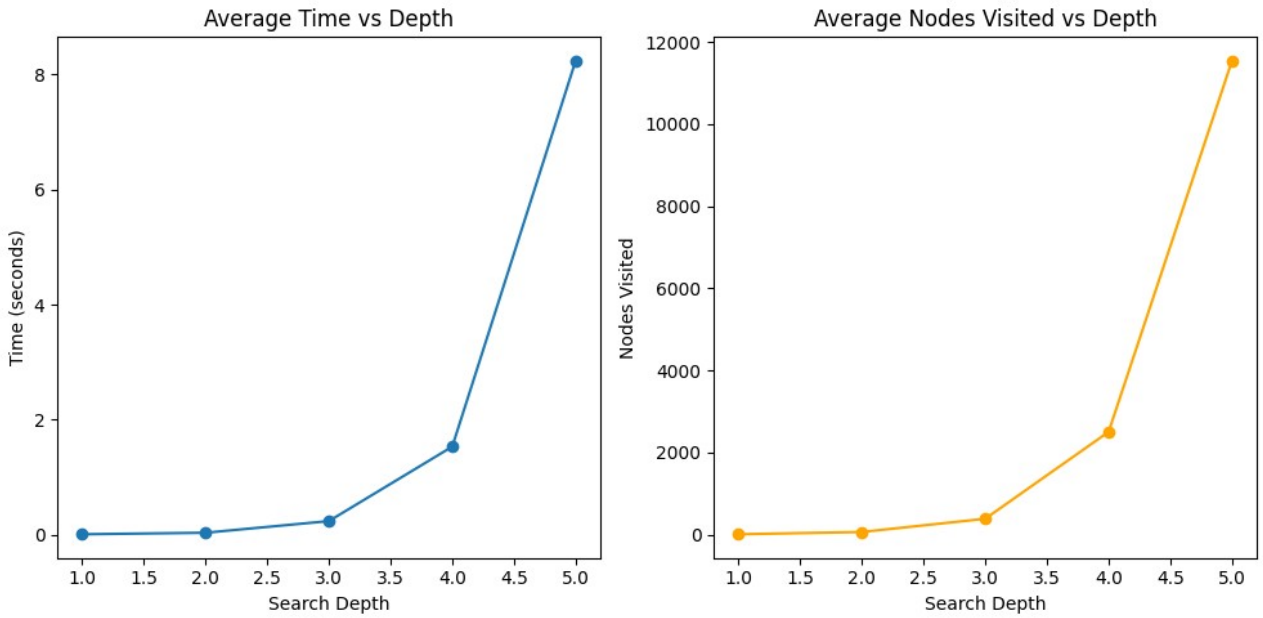


*Figure 2: The graph above illustrates the average time taken and nodes visited by the minimax algorithm with alpha-beta pruning for different depth cutoffs (1-5).*

## Nodes visited:

Without alpha-beta pruning, the number of visited states grows exponentially as the depth increases, reaching about **10,000 states** at depth 4 and **over 50,000 states** at depth 5. This is expected because plain minimax explores all possible states exhaustively, leading to rapid growth in the state space illustrated by the following formula:

$$N = b^d$$

Where $b$ is the branching factor (the average number of legal moves from each position) and $d$ is the depth of the search.

When alpha-beta pruning is applied, the number of visited states significantly decreases, especially as the depth increases. At depth 5, for example, the number of visited states is reduced to **around 12,000**, compared to the 50,000 states in plain minimax. This reduction is due to the alpha-beta pruning process, which cuts off large parts of the search tree when it determines that a move cannot lead to a better outcome than already considered moves.

## Time

For **plain minimax**, the time required increases exponentially as the depth increases. This is expected, as the algorithm must explore the entire game tree, which grows exponentially with depth. It has a time complexity of O(bd), where b is the branching factor and d is the depth.

At a depth of 5, the plain minimax algorithm took approximately **35 seconds**, whereas with **alpha-beta pruning**, the same depth required only **8 seconds** thanks to the pruning process reducing the number of nodes the algorithm needs to explore. Alpha-beta pruning reduces the time complexity to O(bd/2) in the best case, doubling the depth the algorithm can explore within the same time.

# Does the number of states visited depend on the order in which you generate new states during the search?

The number of states visited **does depend** on the order in which you generate new states during the search when using **alpha-beta pruning**.

**For plain minimax**, the order of generating new states has **no impact** on the number of visited states because the algorithm explores every single possible move at each depth, regardless of which moves are considered first. Every branch of the game tree is fully expanded, meaning all states are visited.

**With alpha-beta pruning**, however, the order of state generation determines the number of states visited. If the algorithm encounters a strong move (one that maximizes or minimizes the outcome effectively) early on, it can prune more area of the tree. Conversely, if weaker moves are explored first, the algorithm has less information to prune, resulting in more visited states before it can perform pruning.

## Result from the Power-Connect 4 algorithm

When beginning the game on the left side of the board by dropping a piece on column 1 and exploring the state tree from left to right, the algorithm evaluates high-scoring moves first because pieces will aggregate on the left side, offering opportunities for runs or blocking the opponent, which allows for faster pruning. This focused search results in fewer nodes being visited, leading to quicker evaluations.

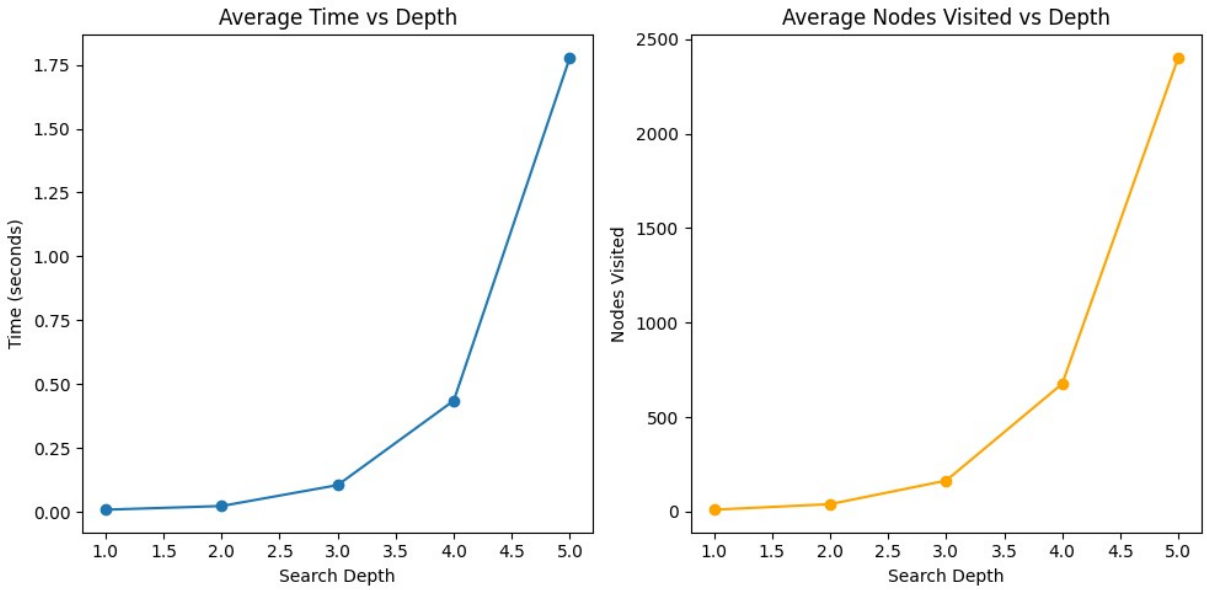


*Figure 3: The above graph shows the average time taken and nodes visited by the algorithm at different search depth for a "left to right" exploration of the tree with a first piece dropped on column 1 "D 1"*

In contrast, when moves are **randomized**, the algorithm tends to explore a wider range of less promising moves first. It is in accordance with observation, both the time taken and the average number of nodes visited roughly doubled compared to the ordered evaluations.

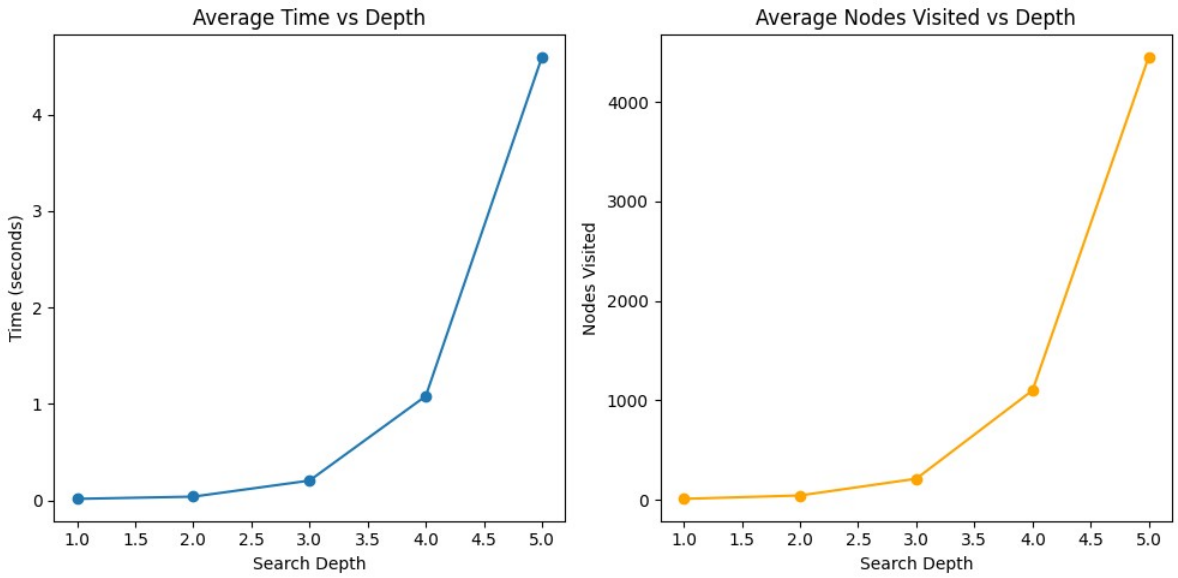


*Figure 4: The above graph shows the average time taken and nodes visited by the algorithm at different search depth for a randomized exploration of the tree with a first piece dropped on column 1 "D 1".*

Conversely, when starting the game with a piece dropped on column 8 on the right side of the board the "left-to-right" move evaluation yield slower results compared to random search which remain stable, see the graphs below

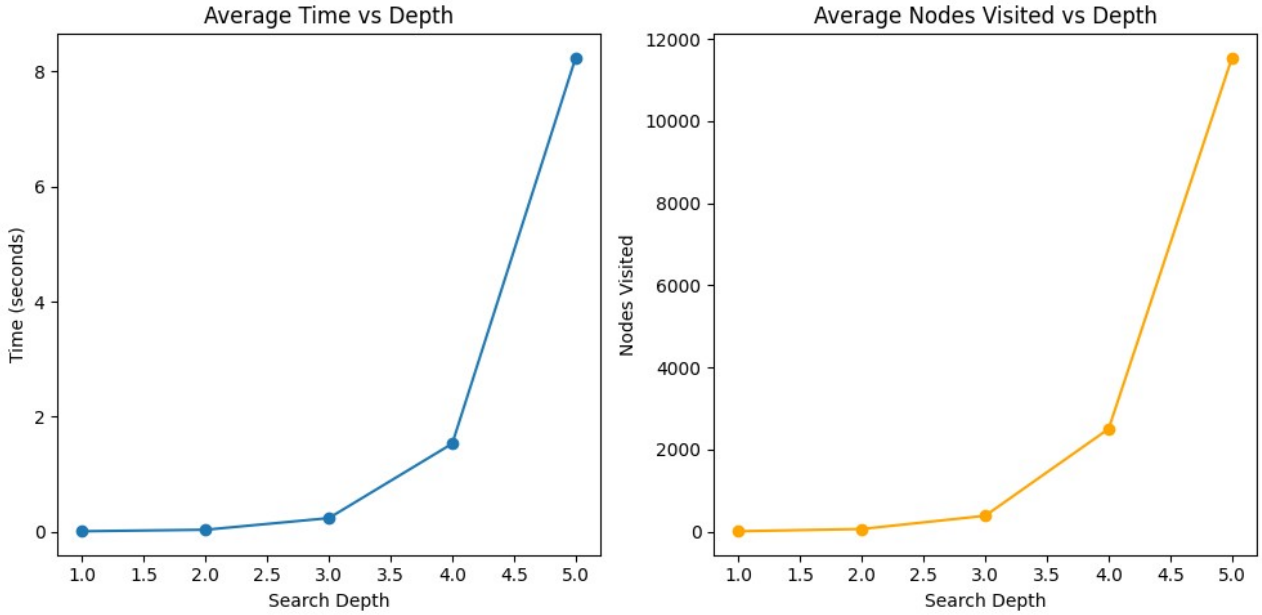


*Figure 5: The above graph shows the average time taken and nodes visited by the algorithm at different search depth for a "left to right" exploration of the tree with a first piece dropped on column 8 "D 8"*

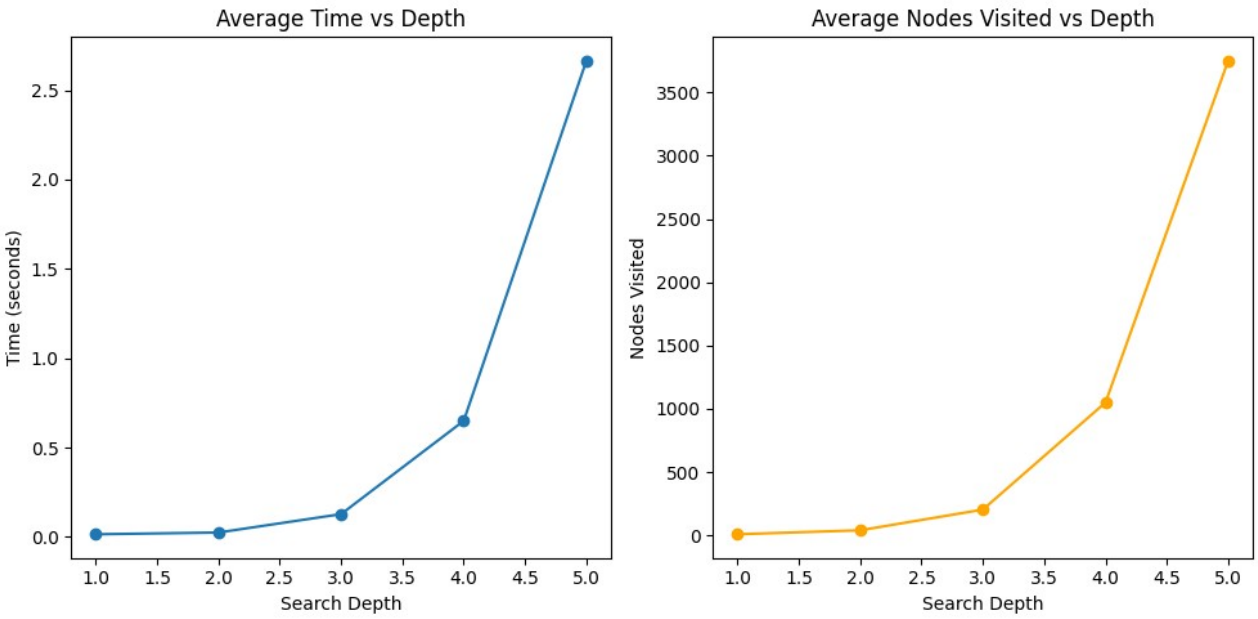


*Figure 6: The above graph shows the average time taken and nodes visited by the algorithm at different search depth for a "randomized" exploration of the tree with a first piece dropped on column 8 "D 8"*

Depending on the starting position of the board, ordered moves can yield significantly different results in terms of computation time and state space extension. Randomized move selection is more table across starting board configuration but is not particularly optimized. An algorithm that can order strong moves first will be more efficient for alpha-beta pruning (it has been tried but I noticed that I had to search through (depth - 1) compared to the condition with randomized ordering.

# Part II

After playing as a human against the AI several time, I tried to identify which board positions were advantageous and why. I came to the conclusion that I used a set of different factors to weigh the quality of the board when it is my turn to play:

1. **Runs of 2 or more pieces:** This was already implemented with the naive heuristic given in the instructions, I must admit that when playing at a beginner level such as mine, it is the main heuristic used and perform well to not lose against an intermediate opponent (my roommate).
2. **Potential for Future Moves**: Evaluate open-ended runs (e.g., a run of 3 with empty spaces on both ends) higher than closed runs, because fully closed runs are not very interesting especially when a significant portion of the board is stacked with 2 or 3 runs that cannot be moved.
3. **Center Control**: According to an advanced guide of the traditional Connect-4 game (https://papergames.io/docs/game-guides/connect4/advanced-guide/) the control of the center columns is beneficial. To create a "score" for the center columns I manually entered notes to each position of the board that I valued while playing, and tried iteratively to feed different weights to both agent playing to see which one will beat the other. This is the final "scoring board" I got :

[0, 0, 0, 0, 0, 0, 0, 0],
[0, 3, 3, 3, 3, 3, 3, 0],
[0, 3, 9, 9, 9, 9, 9, 0],
[0, 3, 9, 13, 13, 9, 3, 0],
[0, 3, 9, 13, 13, 9, 3, 0],
[0, 3, 9, 9, 9, 9, 3, 0],
[0, 3, 5, 7, 7, 5, 3, 0],
[0, 0, 0, 5, 5, 0, 0, 0]

4. **Threats**: Assess not only the player's potential winning moves but also the opponent's threats reducing the total score.
5. **Removable opponent pieces:** A distinct feature of this version of Power-Connect four is the ability to slide 2 or 3 pieces and 1 opponent piece, occasionally outside of the board, thus reducing the piece number of the opponent. I figured explicitly implementing it to the heuristic can lead to interesting “aggressive” behavior from the AI

An example of an improved evaluation function might look like this:

$$H(N) = (5 \times (runs\ of\ 3) + 3 \times (runs\ of\ 2) + (open\ squares\ adjacent\ for\ runs\ of\ 2 \wedge 3)) - .$$
$$(4 \times (opponent's\ run\ of\ 3) + 3 \times (opponent's\ run\ of\ 2) + (opponent\ open\ square))$$
$$+ (center\ control\ score) + (removable\ opponent\ piece)$$

The behavior is purposely oriented towards aggressive gameplay by giving a higher factor to one’s own runs compared to opponent’s run.

It must be noted that open square adjacent to runs of 3 are twice as valuable as for 2.

**Computational Trade-offs**

While a more complex heuristic can improve the strategy of the program, it comes with trade-offs in terms of computational efficiency:

1. **Increased Computation Time**: Evaluating a more sophisticated function requires more computational resources. Each call to the evaluation function may take longer, especially if it includes additional logic for counting runs, assessing threats, and calculating center control.
2. **Depth of Search**: With the increased time per evaluation, the depth of the game tree evaluated is lower in a given time frame. For example, the naive heuristic was previously evaluating up to depth 5 within the 10 second limit, using a more complex function might limit it to depth 3 or 4 for the same limit.

The challenge is to find a balance where the evaluation function is intelligent enough to improve decision-making while still being efficient for timely evaluations to maintain a reasonable search depth.

## Comparison of Naive and Improved Heuristic: Depth Cutoff of 4

To assess the performance of the two heuristics, I logged games played by AI agents using a depth cutoff of 4. The WHITE agent used the **naive heuristic** from Part 1, while the BLACK agent used the **improved heuristic** from Part 2. logs of the game are attached to the submission folder, the first turn between player is always random.

### 1. Naive Heuristic

The naive heuristic evaluates board states by simply counting the number of runs of 2 or more pieces for both players. As seen previously, this approach leads to a higher number of visited states, indicating that the AI explores many suboptimal moves. In the log, this is reflected by the agents making several moves without clear long-term planning, resulting in slower game progression.

**Key Observations from the Log**:

- The AI often reacted defensively, focusing on blocking immediate threats instead of building a strong offensive strategy.
- The time per move is higher, as the AI evaluates many unnecessary states before selecting a move.
- The log indicates a lack of focus on center control, with the AI placing pieces on the edges.

### 2. Improved Heuristic

The improved heuristic assigns higher weights to strategic factors, including runs of 3, runs of 2, and center control, while penalizing the opponent's advantageous runs.

**Key Observations from the Log**:

- The AI consistently prioritized central columns and made aggressive moves that built runs of 3 or more pieces.
- By blocking the opponent’s longer runs while extending its own, the AI reached a winning position by turn 15.
- The improved heuristic results in a more efficient and decisive game, as reflected in both the game log and the reduced number of visited states.

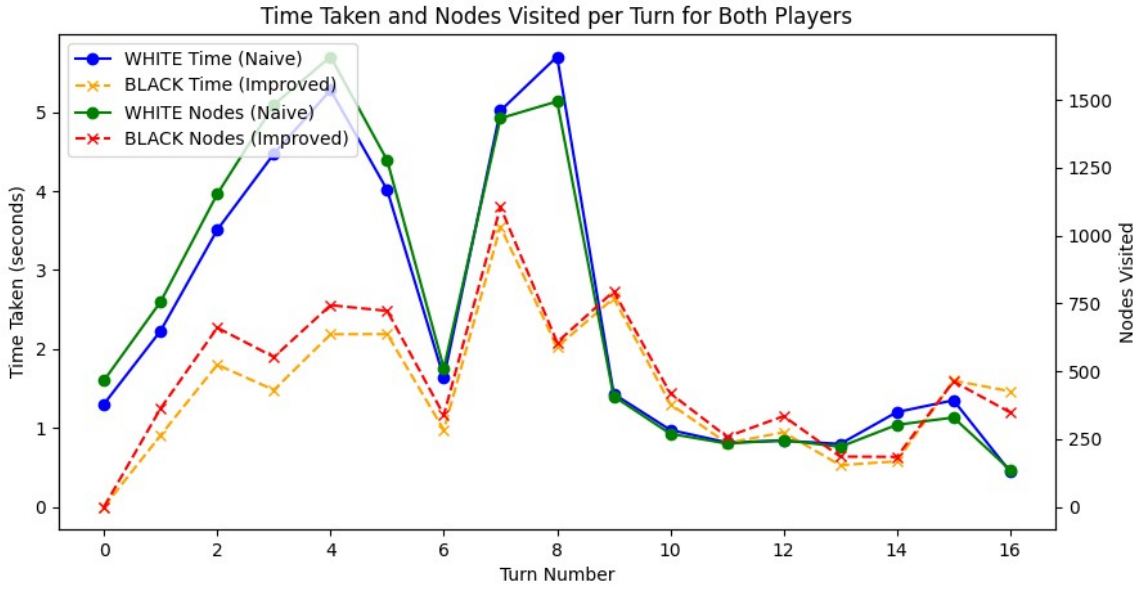


*Figure 7: The graph above shows the time taken and nodes visited for each turn of the match with a depth of 4.*

The improved heuristic is on average quicker and require less node visited to perform a move against the naive heuristic.

## Influence on the Average Number of Nodes Visited for a set depth

This is how I order the following factors :

1. **Use of Alpha-Beta Pruning over minimax:**Alpha-beta pruning will reduce the number of nodes explored most of the time, but it is strongly dependent on the order of child node visits as demonstrated in Part 1.
2. **Order of child node visits**
3. **Choice of Evaluation Function:** The evaluation function plays a crucial role in determining the quality of the node's value assessment. If the evaluation function differentiates well between good and bad states, it can lead to more effective pruning. A well-designed heuristic can result in a significant reduction in the number of nodes visited, as the algorithm can directly disregard suboptimal branches early in the search.

### Guaranteed Winning Sequences in Other Games

For more complex two-player competitive games than connect 4 like chess and checkers, the situation can be different with no guaranteed winning sequence of play independent of the actions of the opponent or not computationally achievable.

**Checkers** is roughly one million times as complex as Connect Four (Schaeffer et al., 2007) with numerous possible moves and a much larger search space. The game of checkers has been solved with Chinook algorithm (https://www.nytimes.com/2007/07/19/science/19cnd-checkers.html) in 2004. It used alpha-beta search algorithm and Df-pn algorithm to efficiently search the game tree.

An experimented player can at best achieve a draw against Chinook. It shows that there is no guaranteed winning sequence. Perfect play from both sides typically leads to a draw, but definitive winning paths have not been established.

**Chess** is even more complex than checkers, with an estimated $10^{120}$ possible game states, far exceeding the complexity of checkers (https://en.wikipedia.org/wiki/Solving_chess). As of today, chess has not been solved, and it is unlikely that a guaranteed winning sequence exists. In practice, perfect play by both sides typically results in a draw. Advances in chess engines, such as Stockfish (https://en.wikipedia.org/wiki/Stockfish_(chess)) and AlphaZero (https://fr.wikipedia.org/wiki/AlphaZero), which use deep search techniques and neural networks, have greatly improved the understanding of optimal play, but the game remains unsolved. Thus, like checkers, chess does not appear to have a guaranteed winning sequence, though it is possible that future advancements could bring the game closer to being solved.

# References

Jonathan Schaeffer *et al.*, Checkers Is Solved.*Science* **317**,1518-1522(2007).
DOI:10.1126/science.1144079