Scientific Research Publishing

Applied Mathematics, 2023, 14, 419-427
https://www.scirp.org/journal/am
ISSN Online: 2152-7393
ISSN Print: 2152-7385

Evaluation of the Use of Minimax Search in Connect-4

—How Does the Minimax Search Algorithm Perform in Connect-4 with Increasing Grid Sizes?

Abdoul Wahab Touré

Department of Mathematics, Le Collège Bilingue, Dakar, Sénégal
Email: awtcanada@gmail.com

How to cite this paper: Touré, A.W. (2023) Evaluation of the Use of Minimax Search in Connect-4. *Applied Mathematics*, **14**, 419-427.
https://doi.org/10.4236/am.2023.146025

Received: May 1, 2023
Accepted: June 10, 2023
Published: June 13, 2023

Abstract

As computers have become faster at performing computations over the decades, algorithms to play games have also become more efficient. This research paper seeks to see how the performance of the Minimax search evolves on increasing Connect-4 grid sizes. The objective of this study is to evaluate the effectiveness of the Minimax search algorithm in making optimal moves under different circumstances and to understand how well the algorithm scales. To answer this question we tested and analyzed the algorithm several times on different grid sizes with a time limit to see its performance as the complexity increases, we also looked for the average search depth for each grid size. The obtained results show that despite larger grid sizes, the Minimax search algorithm stays relatively consistent in terms of performance.

Keywords

Minimax, Alpha-Beta Pruning, Connect-4, Algorithms

1. Introduction

Search algorithms for playing or even solving board games have been used and developed since the invention of computers. Many board games are complex and decisive problems where an action in the present has subsequent outcomes when the game ends in victory or defeat. Because these board games have so many outcomes, it is very difficult for humans to visualize and master the game in a short time [1]. Search algorithms are an important part of game algorithms and are often used in combination with other AI techniques to create powerful and efficient game programs such as Stockfish, which is the most powerful chess

engine at the time of this writing. For simpler games like Tic-Tac-Toe, a computer can easily search for all future possibilities and still be able to make the best move. However, for most games like chess or go[1], the state space is far too large to be traversed by brute force, and the question of playing these games then becomes how to play them efficiently while respecting the restrictions on search time [2], and for this research paper, we will see how that applies to Connect-4. Connect-4 has been chosen because it has moderate complexity and increasing its grid size can greatly affect how an algorithm performs. Knowing this we will see how an increasing grid size correlates with the performance of the Minimax search algorithm. We will first understand how Connect-4 works in the context of game theory, then we will see how the Minimax search algorithm and its different optimizations are used to make it more performant, and finally, we will analyze and discuss the results of our experiments.

2. Related Works

A notable work related to this research paper is "Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game" by Xiyu Kang, Yiqi Wang, and Yanrui Hu [3] in which they go into depth about how the Minimax search algorithm works in Connect-4 and the use of different heuristic functions to improve the algorithm. Another research paper that goes into depth about the Minimax search algorithm is "Alpha-Beta Pruning in Mini-Max Algorithm-An Optimized Approach for a Connect-4 Game" by Rijul Nasa, Rishabh Didwania, Shubhranil Maji, and Vipul Kumar4 [4] where they evaluated to what extent Alpha-Beta pruning, which we will use in this research paper, increases the performance of a Minimax search algorithm in Connect-4. There have also been similar research papers comparing Minimax to other search algorithms such as "Solving Connect 4 Using Optimized Minimax and Monte Carlo Tree Search" by Kavita Sheoran *et al.* [1] where they compared the Minimax algorithm and Monte Carlo Tree Search that both have different approaches to finding optimal moves in zero-sum games.

3. Theoretical Context

3.1. Connect-4

Connect-4 is presented in the form of a game board, with 7 columns and 6 rows where chips of different colors are stacked. Two players compete with the objective of aligning 4 chips of the same color, horizontally, vertically, or diagonally as in Tic-Tac-Toe [5]. Each player slides his or her token in succession so that the game is a zero-sum game, which is a term used in game theory in which one person's gain is equivalent to another's loss. The Connect-4 game grid can be represented as a matrix and the chips can be represented by different values, such as 0 being an empty slot, 1 being the first player and 2 being the second

[1]Go is an abstract strategy board game for two players in which the goal is to surround more territory than the opponent. The game was invented in China over 2500 years ago and is believed to be the oldest board game still played today.

player (see Figure 1). This matrix representation of the Connect-4 grid with a 2D table allows the computer to easily play moves, analyze the grid and check for winning moves.

About 4.5 trillion board configurations are possible for Puisssance 4, which classifies it as a moderate complexity game [1]. When we will need to increase the size of the grid for the experiment, we will choose dimensions of $(n+1)\times n$ with n being an even number to have odd numbers of columns and even numbers of rows which refers to the Connect-4 fundamentals where there is a central column.

$$A_{7\times 6} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 2 \\ 2 & 1 & 0 & 1 & 1 & 2 & 1 \\ 1 & 2 & 0 & 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 1 & 1 & 1 \end{bmatrix}$$

Figure 1. Matrix representation of a Connect-4 game where player 2 won by aligning four "2" diagonally.

3.2. Minimax Search

Let's start by explaining the Minimax search algorithm because it is the simpler of the two algorithms in this experiment. The Minimax search algorithm is a backtracking algorithm[2] used in decision making, game theory, and artificial intelligence (AI). It is used to find the optimal move for a player, assuming that the opponent also plays optimally [6]. Its name comes from its objective to minimize the opponent's score while maximizing our own score with each move made [7] because our opponent will constantly try to minimize our chances and we will constantly try to maximize our chances, which brings us back to the zero-sum nature of Connect-4 in which one's gain is always the opponent's loss and vice versa. Minimax can be represented by a game tree where each node represents the value of a game state starting from the root node (level 0) which is the initial game state, branching into "child" nodes (level 1) which will branch into "grandchild" nodes (level 2) and so on until reaching a terminal state [8] (see Figure 2). Knowing that a standard Connect-4 set can have up to 4.5 trillion combinations, branching all the nodes in a Minimax search algorithm would be an impossible task for modern machines to compute in a reasonable time (see Table 1). For this reason, we must limit our search depth level. In theory, the greater the search depth, the more knowledge the Minimax search algorithm has to find the best move. With this information, we need to have a search depth that is not so large that it takes too long to compute, but not so small that the search algorithm has very little information.

In Figure 2 below, we can see that the minimizing nodes took the smallest

[2]A backtracking algorithm is a problem-solving algorithm that uses a brute force approach to find the desired result.

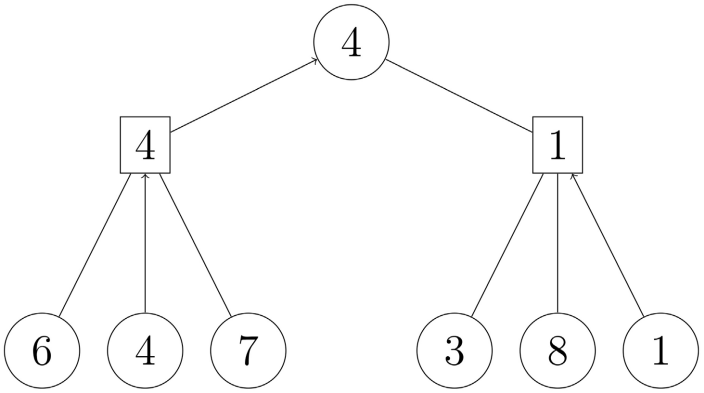


Figure 2. Tree representation of a Minimax algorithm where the circle represents the player who maximizes and the square the one who minimizes.

Table 1. Number of legal Connect-4 positions after n depth levels.

Search depth	Number of legal positions
n	B
0	1
1	7
2	49
3	238
4	1120
5	4263
6	16,422
7	54,859
8	184,275

values among their children, 4 and 1 respectively, and that the maximizing node (squares) at the top which is also the root node (circle) took the largest value among its two children, in this case, it chose 4. We see that the Minimax search tree is a recursive algorithm in the sense that the maximizing player calls the minimizing player, and the minimizing player calls the maximizing player, this assumes that both players will always play the most advantageous move when it is their turn. The conditions for exiting the recursive loop are: if we reach a node where someone has won; if the two players have tied; or if we have reached a predetermined depth limit [8].

The time complexity of a Minimax search algorithm is $\mathcal{O}\left(b^m\right)$ where b is the branching factor or the number of legal moves at each point and m is the maximum depth of the tree. This time complexity is moderate for a typical 7 × 6 grid, but as the grid size increases, the effects of time complexity become clearer as the maximum depth of increasing grid sizes is larger and the branching factor also becomes larger. To alleviate this problem, we will need to significantly reduce the number of nodes to be searched, and fortunately, there is an optimization of

the Minimax search algorithm called Alpha-Beta pruning that we will discuss in the next subsection.

3.2.1. Alpha-Beta Pruning

Alpha-beta pruning is an optimization of the Minimax search algorithm. Before we dive into the alpha-beta pruning algorithm, let's define what "pruning" means in alpha-beta pruning. The word "pruning" means cutting off branches and leaves. Thus, alpha-beta pruning is nothing more than pruning unnecessary branches in decision trees [9], in this case, pruning unnecessary branches in a Minimax search algorithm. As we said in the Minimax section, the search tree becomes extremely complex as the search depth increases, some unnecessary branches in this tree add to the complexity of the search algorithm. Alpha-beta pruning removes these branches, saving the computer from examining the entire tree [9]. In order to know which branches of the search algorithm to prune, we need two threshold parameters: α (alpha) and β (beta). These values represent the worst-case scenario for each player. The value of α is initially set to -∞ and will be updated to **the highest value** each time it is the maximizing player's turn and the value of β is initially set to ∞ and will be updated to **the lowest value** each time it is the minimizer's turn. The condition for pruning a branch or a subtree[3] is when $\alpha \geq \beta$, because all the higher values that α can have in the sub-tree of the maximizing player will be useless because the minimizing player has already found a move that would be more advantageous for him, it also works in the other way, all the lower values that β can have in the sub-tree of the minimizing player will be useless because the maximizing player has already found a move that would be more advantageous for him. It is important to note that the values of α and β are passed on during the backtracking because the Minimax search algorithm uses the depth search to traverse the tree (see **Figure 3**).

In **Figure 3**, we can see that this is the same Minimax search tree that we had in the previous section, but this time the second node of the minimizer has chosen 3 instead of 1, pruning the other two nodes because even though the minimizer is finding values less than 3, the maximizer has already found a value 4

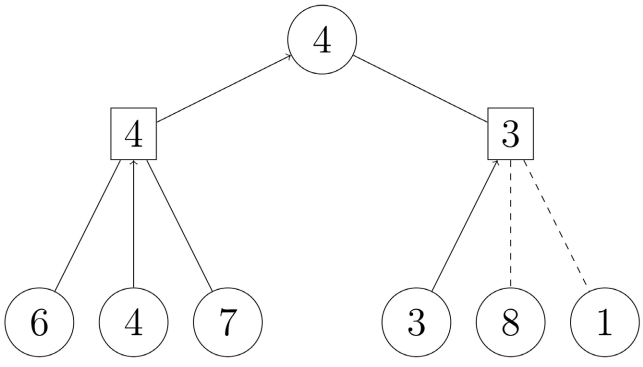


Figure 3. Same Minimax search tree in **Figure 2** but with Alpha-beta pruning implemented and two nodes that have been pruned.

[3]A subtree is a subset of a larger tree containing branches.

that is greater than 3, so an additional search in the minimizer's path would be a waste of time.

The amount of branches pruned by alpha-beta pruning depends on the order of the values. The worst order occurs when the best values for the maximizing and minimizing player are at the end, which means that the best values are on the right side of the tree, because of this Minimax with $\alpha\beta$ pruning[4] behaves like a normal Minimax search algorithm by searching all nodes, where the time complexity is $\mathcal{O}\left(b^m\right)$. The ideal order, which means that the best values are always the first ones to appear on the left side of the tree, in this optimal case, Minimax with $\alpha\beta$ pruning will have a time complexity of $\mathcal{O}\left(b^{m/2}\right)$. The reason it is $\mathcal{O}\left(b^{m/2}\right)$ in an ideal order scenario is that essentially all the values of the first player must be examined to find the best one, but for each, only the second player's best move is necessary to refute all but the first (and best) move of the player [10], in other words, Minimax with $\alpha\beta$ pruning will skip every other level. In almost all Minimax search trees with $\alpha\beta$ pruning, the order of the values will be neither the worst nor the ideal but somewhere in the middle (see **Table 2**). We could order the movements (values) to be ideal using various heuristic functions, but that is beyond the scope of this research paper.

We can see in **Table 2** the Minimax search algorithm with $\alpha\beta$ pruning, even without ideal order, traverses far fewer nodes than a standard Minimax search algorithm as the depth level (ply) increases in Connect-4. With this information, we can assume that for the same number of nodes traversed, the Minimax with $\alpha\beta$ can go into a much larger depth level than the normal Minimax.

Table 2. Table of the number of nodes traversed at different search depths (ply) for Minimax and Minimax $\alpha\beta$ at Connect-4 (Source: Nasa, Rijul et Didwania, Rishabh et Maji, Shubhranil et Kumar, Vipul, 2018. [4])

	Minimax	**Minimax + $\alpha\beta$**
ply 1 (easy)	7	7
ply 4 (medium)	2799	477
ply 8 (difficult)	5,847,005	71,773

3.2.2. Evaluation Function

In the Minimax section and the alpha-beta pruning subsection, the search trees had values in which these algorithms worked to have a result, but where do we actually get these values in a Connect-4 game? To deal with these values, we need an evaluation function that evaluates the state of a set[5] after a player's move and assigns it an appropriate value that the maximizing and minimizing players can use to find the best moves. Essentially, each node in the search tree is the value of a different board state.

The evaluation functions are game specific and can be tuned. For this experiment, we will use a simple Connect-4 evaluation function. This evaluation func-

[4]Shorter expression for alpha-beta pruning.

[5]A board state is a board configuration of different moves/movements.

tion assigns a weight to all groups of chips[6] depending on how many chips are lined up in a group, for each player. The value that will be returned by the evaluation function is the sum of all weights of the maximizing player minus the sum of all weights of the minimizing player (see Equation (1.1)).

$$\text{Eval} = \sum_{i=0}^{n} w_i - \sum_{i=0}^{n'} w_i \tag{1.1}$$

- n = number of token groups for the player who maximizes
- n' = number of groups of chips for the player who minimizes
- i = the index of a group of chips
- w = weight assigned to a group of tokens (see Table 3)

The reason these weights were chosen is to emphasize the groups of chips that have more aligned chips. As we see in Table 3, for two lined-up chips we give it a score of only 1, but for three lined-up chips, we give it a score of 10 because it may be one chip away from winning, and finally for four lined up chips we give it a score of 1000 because it is a winning state which is the only thing that matters to us. Having weights for a higher number of lined-up chips would be useless because the number of lined-up chips needed to win is only four.

Table 3. Associated weight for each number of aligned chips.

Number of tokens lined up	Weight (w)
2	1
3	10
4	1000

4. Methdology of the Experiment

For the experiment, I tested the Minimax search algorithm with Alpha-Beta pruning against a greedy agent[7] 100 times (50 times as the first player, 50 times as the second player to give each player the chance of having the first move) with a time limit of 1 second, for a 7 × 6 board, 9 × 8 board, 11 × 10 board, and a 13 × 12 board. These board sizes have been chosen because like we said previously, we will choose dimensions of $(n+1)\times n$ with n being an even number to have odd numbers of columns and even numbers of rows which ties back to the Connect-4 fundamentals. This experiment has been conducted with the usage of the Kaggle Connect-X [11] module which helps us run our algorithms without the need to adapt them for each grid size. For our win rate metric, we will consider a win as 1 point, a draw as 0.5 points, and a loss as 0 points. We will also calculate the average search depth of our algorithm to see if there is any correlation between the search depth and the size of our grids. We used the Kaggle kernel to conduct the experiments so the processor used is an Intel(R) Xeon(R)

[6]A group of chips consists of two or more chips of the same color that are lined up vertically, horizontally, or diagonally.

[7]An agent that will always take the winning move if available, the agent will also observe if the other opponent can win of the next move and block accordingly.

CPU @ 2.30 GHz with 4 GB of RAM.

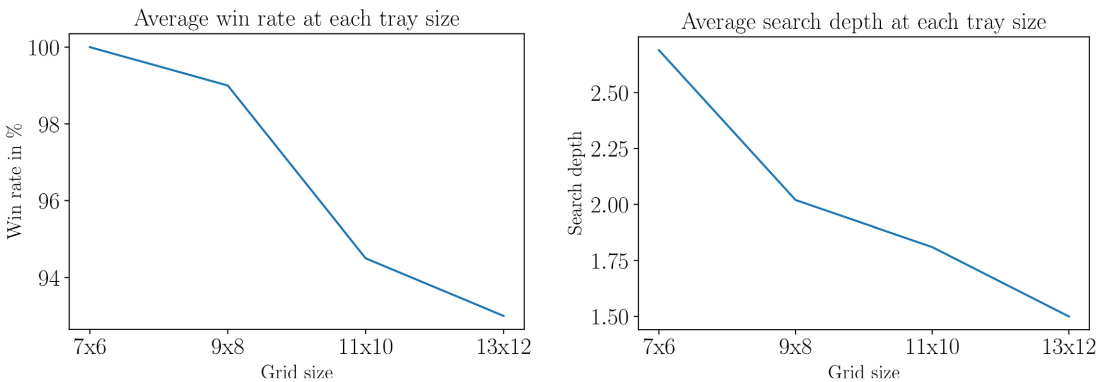
5. Results of the Experiment

Results Analysis

In Table 4 and Figure 4 we can immediately see that as the grid size increases, the overall win rate and average search depth decrease too, this can be explained by the state space increasing when we have bigger grid sizes. Indeed, the more possibilities we have the longer the search will need to be to find the most optimal move. We can also see in Table 4 that the win rate % when playing first is slightly above the win rate % when playing second because the Minimax search algorithm tends to always choose the middle column because the middle is connected to a higher proportion of possible winning spaces. Contrary to our expectations, the win rate at each grid size isn't decreasing at much as we thought so we can conclude that the Minimax + $\alpha\beta$ search is entirely scalable. We can also see in Table 4 that the highest search depth achieved at the 7×6 is very big (21) compared to its successors and that it can be due to a lower state space in the endgames where there aren't many possible moves left for both players because the branching factor is quite low making it easy to search deeper.

Table 4. Minimax + $\alpha\beta$ results table.

Grid size	Winrate % as first player	Winrate % as second player	Overall win rate %	Average search depth	Highest search depth reached
7×6	100%	100%	100%	2.69	21
9×8	100%	98%	99%	2.02	6
11×10	96%	93%	94.5%	1.81	3
13×12	94%	92%	93%	1.50	2



(a) Overwall winrate (b) Average search depth

Figure 4. Graphs of the overall win rate and average search depth for each grid size.

6. Conclusion and Further Research

In this research paper, we compared how the performance of the Minimax search algorithm performance with increasing grid sizes, we then found out how

the performance of the algorithm is related to how deep the search can be. To conclude, we can say that the Minimax search algorithm is efficient at playing Connect-4 with increasing grid sizes because it has a relatively stable win rate with each grid size. We saw that the Minimax search also has a lower search depth average as the grid size increases, slightly impacting its performance as we can see in the graphs of **Figure 4**. This study was limited as we had limited computational power and couldn't do tests with bigger grid sizes as it would be very time-consuming because larger grid sizes tend to have really long games. This experiment was only testing the Minimax search algorithm versus a greedy agent, but testing it against other agents/algorithms might be more indicative of how performant the algorithm truly is. It would also be interesting to integrate reinforcement learning techniques as it could help the search algorithm get more efficient with each game it plays. There are also many heuristic functions we can integrate with Minimax which may help search for optimal moves faster in conjunction with Alpha-Beta pruning. Overall Minimax Search is a simple yet very effective algorithm used in many board games and particularly excels at Connect-4 because of its moderate complexity.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Sheoran, K., *et al.* (2022) Solving Connect 4 Using Optimized Minimax and Monte Carlo Tree Search. Mili Publications. *Advances and Applications in Mathematical Sciences*, **21**, 3303-3313.

[2] Avellan-Hultman, D. and Querat, E.G. (2021) A Comparison of Two Tree-Search Based Algorithms for Playing 3-Dimensional Connect Four.

[3] Kang, X.Y., Wang, Y.Q., Hu, Y.R., *et al.* (2019) Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game. *Journal of Intelligent Learning Systems and Applications*, **11**, 15-31. https://doi.org/10.4236/jilsa.2019.112002

[4] Nasa, R., *et al.* (2018) Alpha-Beta Pruning in Mini-Max Algorithm—An Optimized Approach for a Connect-4 Game. *International Research Journal of Engineering and Technology*, **5**, 1637-1641.

[5] Mf (2021, April) Application of Mcts within the Connect-4 Game.

[6] Vadapalli, P. (2022, October) Min Max Algorithm in AI: Components, Properties, Advantages & Limitations. https://www.upgrad.com/blog/min-max-algorithm-in-ai

[7] Daitzman, S. (2020, December) Minimax. https://mcts.netlify.app/minimax

[8] Eppes, M. (2019, August) Game Theory—The Minimax Algorithm Explained. https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1

[9] Great Learning Team (2022, October) Alpha Beta Pruning in AI. https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai

[10] Wikipedia (2022) Alpha-Beta Pruning—Wikipedia, the Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Alpha%E2%80%93beta%20pruning&oldid=1127344922

[11] Kaggle Connect X. https://www.kaggle.com/c/connectx