Advances and Applications in Mathematical Sciences
Volume 21, Issue 6, April 2022, Pages 3303-3313
© 2022 Mili Publications, India

SOLVING CONNECT 4 USING OPTIMIZED MINIMAX AND MONTE CARLO TREE SEARCH

KAVITA SHEORAN, GEETIKA DHAND, MAYANK DABASZS, NISHTHAVAN DAHIYA and PRATISH PUSHPARAJ*

Department of Computer Science
Maharaja Surajmal Institute of Technology
Janakpuri, New Delhi, India-110058
E-mail: kavita.sheoran@msit.in
geetika.dhand@msit.in
mayankdabas2401@gmail.com
nishthavandahiya7@gmail.com
pratish.pushparaj16@gmail.com

Abstract

Training AI agents for playing and mastering sophisticated board games has come a long way, but it is still a challenging task to excel at. Connect-4 is such a board game with moderate complexity and several trillions of different board configurations. We have trained two other agents in search of a superior agent for playing connect-4 using algorithms "Monte Carlo Tree Search (MCTS)" and Minimax (alpha-beta pruning and dynamic programming), where MCTS and Minimax are game tree search-based algorithms, but we have optimised the Minimax ("alpha-beta" pruning) by using dynamic programming to perform marginally faster, and as a result, it can go deeper in the game tree to give better results. The MCTS agent lost to minimax as because of dynamic programming, minimax could go deeper in the game tree and was able to perform a better tree search.

1. Introduction

Many board games are complex decisive problems where an action in the present results later when the game ends in the form of a win or lose. Now, because all these complex networks of moves lead to a win/lose, it becomes

2020 Mathematics Subject Classification: 68T01.
Keywords: Connect 4, board games, Minima (alpha-beta pruning), dynamic programming, "Monte Carlo Tree Search (MCTS)", artificial intelligence.
*Corresponding author; E-mail: pratish.pushparaj16@gmail.com
Received September 20, 2021; Accepted December 1, 2021

tough for humans to visualise and master the game in a short amount of time. Whereas on the other hand, an AI agent can perform significantly better by visualising future moves or by learning patterns in the games [1].

Connect 4, shown in Figure 1, is a similar board game where winning or losing is dependent on moves performed in the past. This game consists of 2 players and two different types of discs assigned to each player. There is a matrix-like board composed of six rows and seven columns, and each player turn-by-turn put their respective discs in any of the seven columns maximising their chance of winning. Winning is defined as if a player manages to align 4 same types of discs either horizontally, vertically or diagonally, the player wins. Approximately 4.5 trillions board configurations are possible for this game and are categorised as a moderate complexity game.

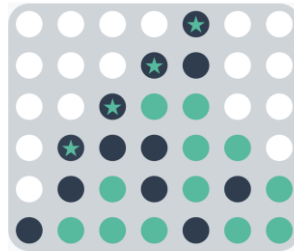

Figure 1. Connect-4 board game.

This paper is an extension of our previous paper [2] in which we trained three different agents in search of a superior agent for playing connect-4 using algorithms Double DQN, “Monte Carlo Tree Search (MCTS)” and Minimax (“alpha-beta” pruning) and we played them against each other. MCTS beat both Double Dqn and Minimax, and further Double Dqn beat minimax. So, Minimax was the worst agent among the three, and MCTS [13] was the best. Now, as we know, the complexity of connect 4 is not very high where we have to choose among seven different moves, the performance of minimax can be significantly improved if we can increase the capability of our agent to look ahead into the future states of our game tree and improve our heuristics for static evaluation of our game state.

So, Our contribution is:

Two different agents are implemented using game tree search-based

algorithms; namely, minimax ("alpha-beta" pruning) "Monte Carlo tree search (MCTS)" to play connect 4.

Minimax is optimised using dynamic programming, and additional advanced evaluation metrics are implemented for better results.

Both the agents compete against each other to choose the most optimal one.

2. Related Works

2.1 Connect 4 using Minimax/MCTS/Double Dqn

This is the previous iteration of this paper [2]. Here, we trained three different agents using algorithms Minimax, "Monte Carlo Tree Search (MCTS)" and Double Dqn (Deep Q learning) to solve connect 4. Minimax and MCTS[15] work by generating a game tree representing current state and future moves and finding the most optimal move by searching the tree and, Double Dqn is a self play algorithm that plays with itself and improves over time by playing numerous episodes of games. Given the same resources all these agents were made to play against each other. our minimax agent was consistently beaten by Double Dqn and MCTS as it was not able to explore higher depth in the game tree and further Double Dqn was beaten by MCTS. So, as a result of our experiments, MCTS was the best and minimax was the worst among the three agents.

2.2 Supervised Connect 4 using Neural Network. Neural Networks have applications in a lot of fields where data is abundant as NNs train on data and learn by adjusting weights. As NNs learn from the data the performance is directly dependent on the quality of data and in case of connect 4 the quality of games saved for training [3]. In this paper, a multilayer artificial neural network [17] model is made with an input layer where board configuration is given as input in the form of 168 neurons as there are 4 options from 1 to 4 for every 42 cells in connect 4 describing empty, my disc, opponent disc, void respectively. Further, a hidden layer with 336 neurons is chosen for better learning and finally an output layer with 7 neurons each denoting the probability of choosing that column. The data is supplied using some explicitly implemented algorithms to play the game. It was also noticed that carefully chosen games were much better training data

for the neural network as expected. Finally, the neural network connect 4 comes out to be on top of all the explicitly programmed algorithms.

3. Methodology

The objective of this paper is to optimize the Minimax algorithm so that the agent could look ahead in more depth at the game tree and improve its performance against agents like MCTS which is one of best performing agents to solve board games. This section provides an overview of the approach to solving the connect-4 using optimized minimax and MCTS and then doing the comparative analysis by making the agents compete against each other. This approach provides us with an effective and efficient way to employ the best model to solve the game.

3.1 Environment. Connect-x Kaggle-environment was used for the episode evaluation (training agents). Kaggle environment [5] was used because it focuses on the episode evaluation, configurable environments, simplified agent and environment creation and cross-language compatibility [6]. All of this made the kaggle environment ideal for training and evaluation of the agents. Also, the Connect-x environment already possessed 2 agents, namely, random (naive agent) and negamax (a variant of minimax) for individual testing of 2 agents before the final comparative analysis. The Nvidia tesla p100 16GB VRAM GPU and 2-core Intel Xeon CPU and 13 GB of RAM were used for training and evaluating performance.

3.2 Methods. The problem was approached by optimizing Minimax using dynamic programming, and "Monte Carlo tree search (MCTS)" which was the best performing agent in our previous paper. Negamax and random agents, that were pre-equipped with the environment, were used for primary testing and then we finally conducted the comparative analysis by making the agents compete against each other. This approach helped us gauge the effectiveness of optimized minimax against MCTS under similar computation constraints as before [2].

3.2.1 Solving Connect 4 with optimized Minimax. The Minimax is an "adversarial search algorithm" [6] responsible for creating the game [12] search tree and helps the agent to look ahead into the future state in the game tree up to depth *'n'* to choose its best move. All nodes in the tree

represent a game state, edges represent different possible moves from a given state, and each level of the tree represents the turn of one of the two players. Two players are denoted by maximizer and minimizer. The maximizer is our agent which aims to maximize its score. The minimizer is our opponent and its goal is to minimize the score of the maximizer, our agent. Once we have exhausted our chosen depth 'n' or if we reach the leaf node, we perform a static evaluation of our node via a heuristic. The heuristic function is different for a different game.

Consider the game tree in Figure 2. For simplicity, we will consider each player to have two possible moves. In the actual game tree, each player has seven possible moves. The depth 'n' is considered as 4 here. Upon depth is exhausted and static evaluation is performed, the score is backtracked to the rooted node of the game tree. The agent does not have complete control of the game. If the agent chooses the left branch, the opponent can force a score of -1 onto the agent. On the contrary, if the agent chooses the right branch, it ends up with at least $a+10$ score with absolute certainty. Hence, the agent is inclined to choose a move that will be propitious for it and opponents will choose a move to reduce our agent's score as much as possible. The opponent plays optimally [4].

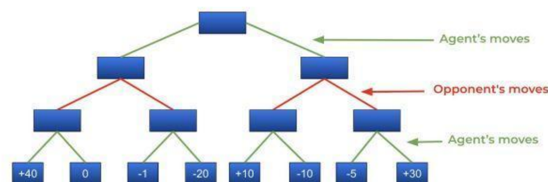

Figure 2. Game tree.

The elementary Minimax does too much unnecessary searching of the game tree; hence mini-max starts to slow up. This is dealt with "alpha-beta" pruning. Applying "alpha-beta" pruning over the minimax [11] algorithm reduces the evaluation of the unnecessary nodes, hence reducing the computation time. The name "alpha-beta pruning" [9] comes from the use of the two parameters, namely, "alpha" and "beta". These two parameters are passed into the function. The agent gets a maximum score stored by the "alpha" parameter, and the opponent tries to minimize the agent's score,

which is stored by the “beta” parameter. To begin with, the “alpha” and “beta” are assigned with the worst possible value for both the agent and the opponent. The algorithm maintains the values of these parameters.

Our previous model couldn’t see the future state of the game beyond the depth of 3. This was due to the computation limitations, and for some nodes, the evaluation function calculated the score more than once. This unnecessary re-evaluation of the same states was dealt with Dynamic Programming. A 2-D array of dimensions 2×5 was created. The rows represent two players. The player at row 0 is our agent and the opponent is at row 1. The index “*j*” ranges from 0 to 4 and starts from $j = 1$. The index “*j*” represents “$j - 1$” aligned discs. The value “table [i] [j]” represents the total count of the number of “*j*” discs aligned by the “ith” player. The discs are counted by creating a window and traversing over the board. Hence, the evaluation function evaluates by taking the values from this table. The agent uses the following scheme [8] for choosing the search depth “*n*”.

$n = 4$ if the empty slots are greater than one-third

$n = 5$ if the empty slots are less than one-third

The heuristic/evaluation function is described below:

Points for 1 disc aligned: table [i] [1]

Points for 2 discs aligned: table [i] [2] * 3

Points for 3 discs aligned: table[i] [3] * 9

Points for 4 discs aligned: table[i] [4] * 81

The opponent also follows the same heuristic but has a negative sign instead of a positive. Hence reducing the score of our agent.

3.2.2 Solving Connect 4 with Monte Carlo Tree Search. “Monte Carlo tree search (MCTS)” [14] is a heuristics-based search algorithm. The heuristics-based search approach makes MCTS an ideal algorithm to solve decision-making board game-like connect-4. The famous “AlphaGo” uses the same MCTS to play “GO” [9]. The major problem with minimax is, it can’t be used for more complex games like ‘GO’ where the branching factor is approximately 300. The MCTS heavily relies on randomness making its evaluation function more reliable for complex games [10].

The principle behind the algorithm [16] is simple: build a search tree node by node based on the simulation playouts. The process can be broken down into 4 sub-processes:

Selection: This sub-process is responsible for selecting the child node based on our upper confidence bound (UCB) value. UCB is calculated using the following formula:

$$UCB = V_i + 2 * \sqrt{\frac{\ln(N)}{n_i}}$$

Where V_i represents the average rewards of all the children of the parent. N represents the number of times the parent is visited and n_i, represents the number of times the, i^{th}, the child node is visited.

Expansion: If the current leaf node is not the terminating node, i.e. the node which represents the end game state, then we expand all of the children of the current node and select one of them based on the UCB value.

Simulation/Rollout: Run a simulated playout, i.e. choose a random move, until we've reached the terminating node. This part is generally performed during exploration.

Backpropagation: Backpropagate the result achieved during the simulation and update the current UCB value.

We put a constraint on our agent to choose its move (denoted by T). This was done to analyze the performance of your MCTS agent [18] against other agents under the effect of the constraint. This gave us the benchmark from where the MCTS was able to outsmart the other agents.

4. Results

4.1. Minimax Vs Random/Negamax

Firstly, the Minimax agent played 100 rounds numerous times each against random and negamax (another variation of minimax) agents which were available in the kaggle environment for individual testing of our agent. Before optimization of minimax, it could not go deeper in the game tree (up to $N = 3$) and hence was not able to get good results against negamax but was

able to defeat random agents in all of the games consistently as it was a naive agent. Now, when the minimax was optimized with dynamic programming it was able to explore deeper in the game tree i.e. $N = 4/5$ depending on the configuration of the board and as a result of that, the optimized minimax improved its performance significantly and beat negamax in 93.75% of the all games played and won against a random agent in all of the games. The winning percentage of minimax against random and negamax is shown in Figure 3.

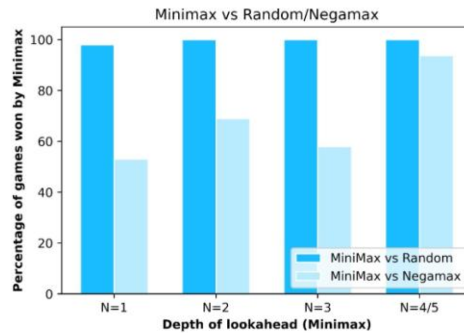

Figure 3. Winning Percentage of Minimax against negamax and minimax agents with increasing depths of lookahead.

4.2 MCTS Vs Random / Negamax. Now, MCTS agent played 100 rounds numerous times each against random and negamax (another variation of minimax) agents for individual testing of this agent. There was a time limit (in milliseconds) for MCTS to choose its move. So MCTS was tested on different time limits and as expected, as the time limit to choose the move increases the winning accuracy of our agent against negamax and random agents increases as well. Figure 4 shows the winning percentages of "MCTS vs random and negamax" with varying time limits.

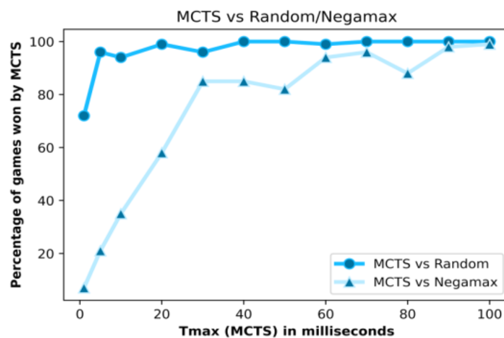

Figure 4. Winning percentage of "MCTS vs negamax and random agents" with increasing $T_{\max}(ms)$.

4.3 Minimax Vs MCTS. Finally, Minimax is put to test against MCTS. 250 games were played at each time limit (Tmax) of "MCTS" with all depths of minimax i.e. $(N = 1, 2, 3, 4/5)$ for generalizing the results. Now, when the allowed time limit (Tmax) is less to choose a move for MCTS, Minimax even at a lower level of depths beats it easily but as the time limit for MCTS increases it starts to show its dominance over minimax upto only $N = 3$ depth but, as the optimized minimax can also go till $N = 4/5$ and because of that even at larger (Tmax) minimax shows exemplary results as compared to other depths i.e. $N = (1, 2, 3)$ with an average win rate of 79%. Figure 5 shows the winning percentage of Minimax with MCTS with various depths and varying Tmax.

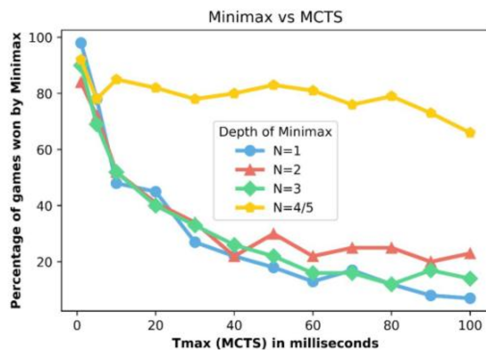

Figure 5. Winning percentage of Minimax against MCTS with increasing $T_{\max}(ms)$.

5. Conclusion

As connect 4 is not a very complex paper, algorithms like minimax should play the game efficiently and effectively. Due to lack of computation power our minimax agent was not able to look ahead beyond the depth of 3 as a result it was not able to win against MCTS. This problem was tackled using dynamic programming and tweaking the heuristics and then we made it play against MCTS which was the best performing agent from our previous paper [2] and was able to defeat both minimax and Double DQN.

As we can see in Figure 5 in the results section MCTS was able to easily beat Minimax up to the depth $n = 3$ given sufficiently enough time to choose its moves. But, once we optimized it using dynamic programming the algorithm was able to look ahead up to the depth of $n = 5$. As a result, minimax showed excellent results against MCTS by maintaining an average winning rate of 79%. Minimax was able to outperform MCTS because minimax was able to look more deeply into the game tree and was able to choose winning moves with surety. On the other hand, MCTS introduced a little bit of randomness in the algorithm to balance the exploration-exploitation trade-off. Although this trade-off and randomness perform an important role in more complex games, like GO, Monopoly etc, for simple games like connect 4, chess etc, it proved as a disadvantage against optimized minimax.

Hence to conclude, if games have low complexity then algorithms like minimax can perform sufficiently well. We can use an optimization paradigm like dynamic programming to improve its capability to look more deeply into the game tree.

6. References

[1] https://www.pnas.org/content/116/30/14785\

[2] M. Dabas, N. Dahiya and P. Pushparaj, Solving Connect 4 Using Artificial Intelligence, Advances In Intelligent Systems And Computing (2021), 727-735.

[3] Marvin Oliver Schneider, João Luís Garcia Rosa: Neural Connect 4 -A Connectionist Approach to the Game, VII Brazilian Symposium on Neural Networks, 2002.

[4] Medium, 2021, Creating the (nearly) perfect connect-four bot with limited move time and file size, [online] Available at:https://towardsdatascience.com/creating-the-perfect-connect-four-ai-bot-c165115557b0[Accessed 17 November 2021]

[5] Connect X. (n.d.), from https://www.kaggle.com/c/connectx.

[6] https://pypi.org/project/kaggle-environments/

[7] Donald E. Knuth Ronald W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6(4) (1975), 293-326.

[8] Xiyu Kang, Yiqi Wang and Yanrui Hu, Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game, Journal of Intelligent Learning Systems and Applications 11(02) (2019).

[9] N. Rijul, D. Rishabh, M. Shubhranil and K. Vipul, Alpha-Beta Pruning in Mini-Max Algorithm- An Optimized Approach for a Connect-4 Game, International Research Journal of Engineering and Technology (IRJET) 5(4) 1637-1641.

[10] Monte Carlo tree search, (2020, September 26), Retrieved December 06, 2020, from https://en.wikipedia.org/wiki/Monte_Carlo_tree_search.

[11] Medium, 2021, Artificial Intelligence at Play—Connect Four (Minimax algorithm explained), [online] Available at: https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f [Accessed 17 November 2021].

[12] [online] Available at: http://www.geeksforgeeks.org/category/algorithm/game-theory/. [Accessed 17 November 2021].

[13] Guillaume Chaslot, Sander Bakkes, Istvan Szita, Pieter Spronck: Monte-Carlo Tree Search: A New Framework for Game AI, Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference.

[14] Monte Real-Time Connect 4 Game Using Artificial Intelligence 1Q learningCarlo tree search, (2020 September 26), Retrieved December 06, 2020, from https://en.wikipedia.org/wiki/Monte_Carlo_tree_search.

[15] https://towardsdatascience.com/game-ais-with-minimax-and-monte-carlo-tree-search-af2a177361b0.

[16] B. Cameron Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton: A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games 4(1) (2012).

[17] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, Humaira Arshad: State-of-the-art in artificial neural network applications: A survey, Heliyon 4(11) (2018).

[18] Prateek Agrawal, Harjeet Kaur, Gurpreet Singh, Indexed Tree Sort: An Approach to Sort Huge Data with Improved Time Complexity, International Journal of Computer Applications (0975-8887), 57(18) (2012), 26-32.