Comparison of Four AI Algorithms in Connect Four

Yiran Qiu[1, †], Zihong Wang[2, *, †], Duo Xu[3, †]
[1]Guangdong Experimental High School, No. 1, Shengshi Road, Liwan District, Guangzhou, P.R.China, 510375
[2]Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180
[3]UIBE, No.10 Huixin Street, Chaoyang District, Beijing, P.R.China, 100029
Corresponding author: wangz37@rpi.edu
[†]These authors contributed equally.

Abstract

It is vital to use machine learning methods to complete game competitions beyond the human level. This paper mainly focuses on the performance of four commonly used algorithms in board game artificial intelligence development, including greedy, alpha-beta pruning, principal variation search, and monte carlo tree search. Specifically, this paper compares algorithms' effectiveness by letting them play connect four against each other 12 times between two methods, alternating the first mover and measuring the time of wins and the total moving steps. Based on wins and total steps of each method, the paper analyzes which algorithm gives the best performance and explains why that algorithm gives the best result. As the result, monte carlo has the highest winning rate against all of the other algorithms with relatively smaller moving steps, and it can win the opponents as fastest compare to other algorithms, which shows that monte carlo gives the best performance in connect four against other algorithms.

1. Introduction

The development of human science and technology consists of the development of games. Early in the Neolithic age, in Northern China, predecessors developed the earliest game. About 3500 B.C., Egyptians invented the earliest board game. Three thousand years ago, in Southeastern Iran, people invented the rolling dice game. People today are still playing these classical games. On the other hand, like the invention of computers, the game has its new form of expression — video games.

Games not only represent an activity that one engages in for amusement or fun, but a particular competition, match, or occasion. In many people's minds, the game is looking easy. The main idea of a game is winning. However, the road to win is much complicated actually. Game is games between players. The Player will predict others' motion and react to others. At any time during the game, players predict and calculate. Take blackjack as an example. The total number of cards in the deck is known. The Player can calculate the probability of drawing a card with a large number. Thus, he would decide to draw or hold. Even in the most straightforward game — rock-paper-scissors, the Player will predict what the opponent will give based on the last round and adjust the strategy in this round[1].

The game we play is full of competition. In past times, humans play against a human. At present, getting benefit from programming languages we can play versus Artificial intelligence (AI). The definition of AI is intelligence demonstrated by machines, as opposed to the natural intelligence displayed by humans or animals[2]. In the early video game, AI is usually set to be the player's opponent and play clumsily, which aims to let the player play the game[3]. However, as time passed, many computer scientists were going to focus on AI's intelligence. They want AI to play as intelligently as humans, even beat humans[4]. Just like Sebastian Thrun said, "Nobody phrases it this way, but I think that artificial intelligence is almost a humanities discipline. It is an attempt to understand human intelligence and human cognition." Enabling AI to think and play like a person will help computer scientists make a more human-like machine.

At the 1956 Dartmouth Conference[5], the term "Artificial intelligence" was confined. In the same year, Arthur Samuel[6] created an AI which plays Checkers using reinforcement learning. In 1958, the first fully functional chess AI was developed. Samuel's Checkers AI won the game against the person in 1962. In 1968, the first Go AI[7] — Zonrist's AI was invented. In 1986, the multi-layer neural network approach was widely known, laying the foundation for a neutral network. 1992 is the milestone year in AI history. Gerald Tesauro wrote TD-Gammon[8], which applied neutral network and TD-Lambda algorithm. It proved that using reinforcement learning without features could build a good AI as clever as humans in playing Backgammon. After that, AIs such as Deep Blue (IBM chess AI beat world champion in 1997), AlphaGo overcome top human players in a Chess game and the game of go. Even in video games[9], alpha star, mainly uses neural network to deal with considered actions such as where to click and what to build, defeated grandmaster human players in the real-time strategy game[10].

These AIs use different algorithms, and all of them do well in a particular game. Our work is to find several different AI which use different algorithms to play Connect Four. Connect Four is an example of an adversarial, finite zero-sum game of perfect information. So, we could apply algorithms that are commonly used in other board games, for example, Greedy, Alpha-beta pruning, Monte Carlo, and Principal Variation Search. For all of these four algorithms, we decide to describe them, talk about their core method, and algorithm in addition to their benefits. We also try to let their AI play against each other about 12 times, alternating the first move each time and recording the winner and the moving steps in total, which will help us analyze the effectiveness of a specific algorithm on Connect Four.

Based on the performance in 36 turns, we look for an algorithm with the best performance in Connect Four compared to other algorithms and try to explain why it has the best performance. After we did the experiment, Monte Carlo gives the best performance according to 30 wins in 36 games and relatively small moving steps. Moreover, the results are consistent with our prediction, which is “MCTS may perform better in the Connect Four because Monte Carlo rollouts allow it to take distant consequences of moves into account”.

2. Method

The rules for Connect Four are simple. The two sides of the game take turns placing pieces, and the pieces fall to the bottom of each column of a vertical board and stack up according to gravity. That is, the bottom of each piece must have "support," not "hanging." It wins when one side has four consecutive pieces in a horizontal, vertical, or oblique line. It is a draw if the board is filled, but there are still no four pieces connected. Connect Four has similarities with FIR and Go. We can compare those values to find the best move if we can express how good a move is in numerical terms. Because Connect Four is so familiar to other zero-sum games, we took algorithms commonly used in other board games -- greedy algorithms, Alpha-beta pruning, PVS, and Monte Carlo Tree Search, and compared them against each other and played a few games against them by human beings. In this way, we attempt to compare the performance of different algorithms in Connect Four. Comparison methods in this paper include the Greedy algorithm, Alpha-beta pruning, PVS, and MCTS.

2.1 Greedy Algorithms

The idea of the greedy algorithm is that in each step required to solve the problem, try to choose the best way to make the current/local state achieve the optimal global state. It tries all the possible moves and checks the static score for each of them. Eventually, it selects the move with the highest score. It does not consider whether current choices make future choices harder, so the decision is not necessarily optimal overall. We chose the greedy algorithm because when humans play Connect Four, they often think about how to achieve a locally optimal solution. Moreover, Connect Four is not a very complex game. Its optimal local solution often represents the optimal global solution. At the same time, the greedy algorithm has very little overhead because it does not have to consider the state of the next few steps.

To calculate the best way to make local states, we set up a formula to calculate the score of the board game, as shown in Table 1. The scoring formula is calculated: We call a string of one or more pieces in a straight-line 'chess string.' One point is scored when its length is 1 (i.e., there is only one piece), two points are scored when its length is 2, four points are scored when its length is 3, and the score is infinite when its length is 4, which is victory. A player's score is the sum of all its chess string scores. By calculating the scores of different positions, the one with the highest score is chosen as the optimal solution. At the same time, we also consider the opponent's piece form. When there are three pieces on the other side, our first choice is to take a blocking step.

Table 1. Scores of chess string

Length	1	2	3	4	
Score	1	2		4	$+\infty$

2.2 Alpha-beta pruning

Essentially, Alpha-beta pruning is a method to search part of the game tree to find the best strategy in this situation and prune some useless tree node to accelerate. Each game tree node has six child nodes that represent each possible situation. In principle, we have to figure out how each of the following moves will affect the game and pick the most advantageous move. In this case, we set the searching depth of the game tree equal to 4. In other words, the algorithm will search four future actions to calculate the best way to move the disc.

Minimax algorithm. Alpha-beta pruning includes the Minimax algorithm, the tree model has two alternate layers, the max layer, and the min layer. The max layer simply means its score is equal to the maxima score of scores of all its child nodes and the min layer means its score is equal to the minima score of all scores of its child nodes.

Core algorithm. The core and difference point of Alpha-beta pruning is its pruning condition: $Alpha \geq Beta$, where Alpha represents the score of max layers, initial in-infinity, and Beta represent the score of min layers., initial in infinity The value of Alpha and Beta is pass from other possible actions in the game tree. Those values are initial in -infinity and infinity which assume they are starting in the worst situation. When the pruning condition, $Alpha \geq Beta$, is true, the node and all its child node is useless or hopeless for us because the value of Alpha and Beta is passed from other situations. In other words, this action is worse than other possible actions. Thus, the algorithm is no need for continuing searching, as shown in Figure 1.

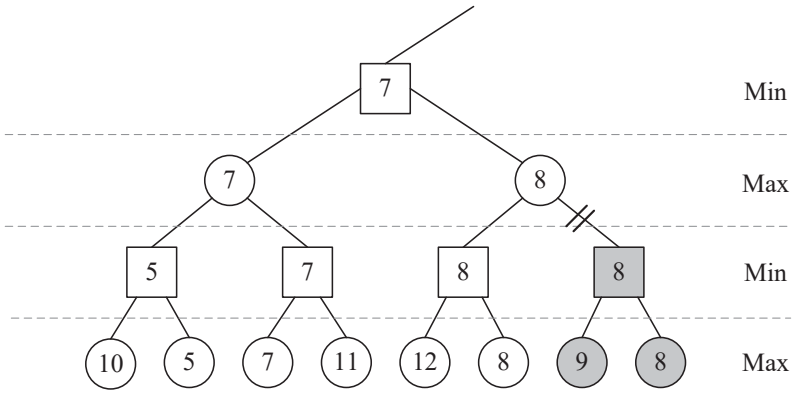

Figure 1 The principle of Alpha-beta pruning

Alpha-beta pruning is a depth-first search algorithm. It uses the maximin algorithm to decrease the number of nodes that should be search or evaluate. This algorithm has better performance in time or space than most of the other algorithms.

2.3 Principal Variation Search

Principal Variation Search (PVS), also known as Minimal Window Search, or NegaScout algorithm. It is based on the basic idea that in a strongly ordered game tree, the first move on each node is likely to be the best. It takes less time to prove that a sub-tree is inferior than to work out the game value of the sub-tree.

Size of the search window. The size of the search window refers to the difference between beta and alpha. The smaller the search window, the greater the probability of pruning.

The core algorithm. PVS starts by assuming that the first move at each node is the best and tries to prove that subsequent moves are relatively inferior until this assumption is proved wrong. The minimum window algorithm uses the entire search window [α, β] for the first subtree of each node to conduct a complete search (exhaustively search) and uses the minimum search window for the remaining subtree of this node to find out whether the subtree is relatively inferior quickly. If it is proved that the current subtree is not inferior, the subtree needs to be searched again, and the game value V needs to be modified. This smallest Window is also called a Null Window.

In the Alpha-beta pruning algorithm, the search process is not complete because of pruning, so the estimation obtained by the pruning algorithm may not be accurate. PVS algorithm is an improvement of the Alpha-beta pruning algorithm, which can improve the efficiency of the Alpha-beta pruning algorithm. When the game tree is strongly ordered, the efficiency of the PVS search is higher.

2.4 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a recently proposed search method that combines the precision of tree search with the generality of random sampling. It does not depend on a heuristic function to evaluate the best next move to make. Instead, it considers the game mechanics to play random rollouts and get an expected reward after a fixed number of iterations. It has been successfully applied to a variety of games, especially board games. MCTS runs multiple game simulations and then tries to predict the best next step based on the simulation results. This is in line with our need for the next step of prediction. However, MCTS builds a highly selective tree and can miss crucial moves and fall into traps in tactical situations. We use a full-width minimax search to avoid this. Monte Carlo rollouts allow it to take distant consequences of moves into account, giving it a strategic advantage in many domains over traditional depth-first minimax search with Alpha-beta pruning. Based on this, we think that Monte Carlo may perform better than Alpha-beta pruning in the Connect Four.

MCTS simulates the game multiple times, along with a set of traversals of the game tree. A single walk is a path from the root node (the current game state) to a not fully expanded node. An incomplete node means that it has at least one child node that has not been accessed. When a node is not fully expanded, one of the node's children has not been accessed is selected for a simulation. The simplest form of simulation is just a random sequence of moves from a given game state to an endpoint. The result of the simulation wins, loses, or draws. The simulation result is then relayed back to the root node of the current tree, and the statistics of the node are updated. The specific process of Monte Carlo is shown in Figure 2.

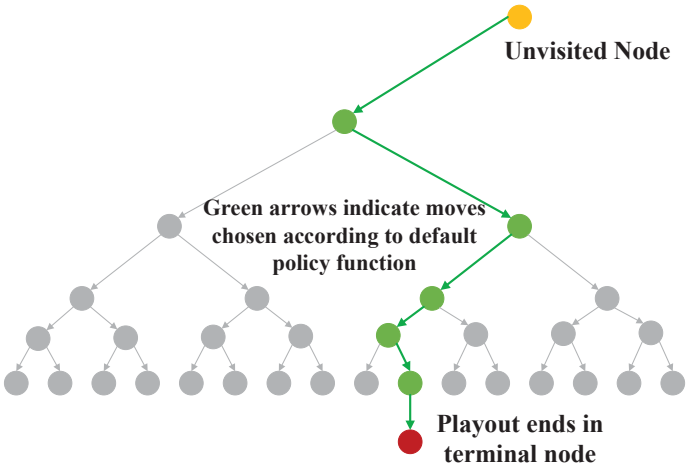

Figure 2. The specific process of MCTS

Core Algorithm. Forward propagation updates the statistics of all nodes along its path, including total simulation revenue $Q(v)$and total access times $N(v)$ as shown in Figure 3. A node's statistics reflect how likely it is to be the best next step. $Q(v)$: In its simplest form is the sum of the simulation results of all the nodes under consideration. $N(v)$: indicates the number of times this node appears on the forward propagation path.

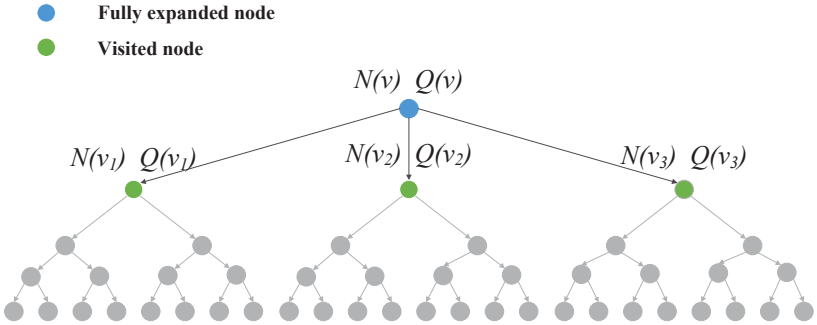

Figure 3 Core algorithm of MCTS

A node is a fully expanded node when all its children are explored. The statistics for this and its children make up our last section: Upper Confidence Bound applied to Trees (UTC). UCT is a function that allows the algorithm to select the next node from the visited nodes to traverse and is also the core function of MCTS. The formula for UCT is shown below.

$$UCT(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log(N(v))}{N(v_i)}} \quad (1)$$

It consists of two parts. The former part $\frac{Q(v_i)}{N(v_i)}$ can be regarded as the win rate estimation of child node VI ($total\ revenue\ /\ total\ times = average\ revenue\ per\ time$). The latter part $c\sqrt{\frac{\log(N(v))}{N(v_i)}}$ prevents the algorithm from discarding the less explored nodes and falling into the local optimal solution.

3. Experimental results and analysis

The purpose of this section is to compare the difference and efficiency of Alpha-beta pruning, Greedy, Monte Carlo, and Principal Variation Search in the Connect Four game. Each experiment is designed as traversing every 12 turns: alternative setting one of two methods as the first mover; and the first mover move its first move from the first column to the last column of the board during six turns. Thus, the complete experiment needs 12 turns. At the same time, we also collect the performance of each method during playing against.

Table 2. Experiment results of Greedy

Opponent name	Win times	Moving steps
Alpha-beta pruning	0	60
Monte Carlo	0	94
Principal Variation Search	0	60
Total	0	214

Table 3. Experiment results of Alpha-beta pruning

Opponent name	Win times	Moving steps
Greedy	12	150
Monte Carlo	1	216
Principal Variation Search	6	252
Total	19	618

ISBN 978-3-8007-5701-9

Table 4. Experiment results of Principal Variation Search

Opponent name	Win times	Moving steps
Greedy	12	150
Alpha-beta pruning	6	252
Monte Carlo	5	235
Total	23	637

Table 5. Experiment results of Monte Carlo

Opponent name	Win times	Moving steps
Greedy	12	96
Alpha-beta pruning	11	154
Principal Variation Search	7	166
Total	30	416

As shown in Table 2, the Greedy method, one of the most straightforward algorithms, has the lowest time complexity and the worst winning rate. Additionally, Greedy has minor moving steps, which means Greedy permanently loses in the early stage. To sum up, Greedy is the simplest but not good at Connect Four algorithm compared to another algorithm. As shown in Table 3, Alpha-beta pruning got 19 winnings during a total of 3 experiments. It ranked third, neither the lowest nor highest time complexity in all algorithms. The moving steps number of this algorithm is the second largest in all algorithms, which means this algorithm does not have a good performance but is not so worse. As in Table 4, the principal Variation Search algorithm has the most significant moving steps and second rank of winning times. This characteristic means the Principal Variation Search always plays with opponents into later stages and wins most of the games but loses some. Thus, the Principal Variation Search does not have the best performance compared to other algorithms, neither in winning time or duration of winning. Table 5 shows that Monte Carlo has the best performance in Connect Four compared to other algorithms. Monte Carlo has the third significant moving step as the most winning time algorithm (30 times in 36 turns). In other words, the Monte Carlo can win the opponents as fastest as other algorithms. Monte Carlo gains the best performance in winning time because it can take distant consequences of moves into account, as discussed in section 2.

4. Conclusions

This article compared four artificial intelligence algorithms commonly used in board games: the Greedy Algorithm, Alpha-beta pruning, Principal Variation Search, and Monte Carlo Tree Search. As can be seen from the final experimental results, the performance rankings of various algorithms in terms of the number of victories are MCTS, PVS, Alpha-beta pruning, and Greedy, respectively. The average number of moving steps required, Greedy, MCTS, Alpha-beta pruning, and PVS, is short to long. In terms of victories alone, THE MCTS algorithm performed best on Connect Four, consistent with the analysis in Section 2. Meanwhile, its moving steps are the least among the four algorithms. In the previous introduction, we learned that the accuracy of MCTS is related to its depth of search. If we want a higher win rate, we need to increase its search depth, and its moving steps will increase accordingly. For fewer moving steps, choose Greedy and MCTS; For a higher win rate, choose PVS and MCTS. However, overall, MCTS performs best among the four algorithms in Connect Four.

References

[1] Yannakakis, G., & Togelius, J. (2018). *Artificial intelligence and games*.

[2] McCarthy, J. What is AI? / Basic Questions. Retrieved 24 August 2021, from http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html

[3] How Artificial Intelligence has Shaped the History of Gaming. (2017). Retrieved 20 August 2021, from https://www.gamedev.net/tutorials/programming/artificial-intelligence/how-artificial-intelligence-has-shaped-the-history-of-gaming-r4782/

[4] History of AI Use in Video Game Design - Big Data Analytics News. (2021). Retrieved 21 August 2021, from https://bigdataanalyticsnews.com/history-of-artificial-intelligence-in-video-games/

[5] Kurenkov, A. (2016). A 'Brief' History of Game AI Up To AlphaGo. Retrieved 21 August 2021, from https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai/

[6] Xu, S. History of AI design in video games and its development in RTS games - Final Step. Retrieved 20 August 2021, from https://sites.google.com/site/myangelcafe/articles/history_

[7] Lee, M. (2015). 11.2 Samuel's Checkers Player. Retrieved 20 August 2021, from http://www.incompleteideas.net/book/ebook/node109.html

[8] Tesauro, G. (1995). *Temporal Difference Learning and TD-Gammon* [Ebook]. Communications of the ACM. Retrieved from https://www.csd.uwo.ca/~xling/cs346a/extra/tdgammon.pdf

[9] Zobrist Hashing. Retrieved 26 August 2021, from https://iq.opengenus.org/zobrist-hashing-game-theory/

[10] AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. (2019). Retrieved 19 August 2021, from https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

[11] Optimal Setting of Weights, Learning Rate, and Gain [J]. Idiap, 1997.