The best AI algorithm defeats the human in board game

Vigneshraj Perumal Raja (a1787474)[1]

[1]School of Electrical and Electronics Engineering, The University of Adelaide, Adelaide, 5005, Australia
Email. Id: a1787474@stude.adelide.edu.au

Abstract:

This paper analysis the method and functions of the existing effective algorithms such as Monte-Carlo process, Minimax Algorithm and Alpha-beta (AB) pruning technique. By evaluating its efficiency and other necessary characteristics, the optimized version of Monte-Carlo algorithm is chosen as the best leading AI algorithm that could help Artificial Intelligence (AI) to defeat the humans in a board game. The nominated algorithm will be used in the second phase of this research project to test different board games.

Keywords: Artificial Intelligence (AI); Board game; Monte-Carlo process; Minimax Algorithm; Alpha-beta (AB) pruning technique

1. Introduction

1.1. Background

Artificial Intelligence is a digital computer stream that deals in understanding the nature of intelligence and produces new intelligent machines or any supporting software that can respond like human intelligence. Games are considered as the elegant method to explain the AI algorithms. Initially, some simple board games like n-puzzle, tic-tac-toe, Gomoku and chess are used in AI courses to trail and implement new algorithms. Later, different board games started to evolve as a mainstream that aids to teach decision-making algorithms in AI [5].

During this time, the AI application of board games was considered as an opponent for the human players. For the first time ever AlphaGo, AI defeated the leading expert of Go in a tournament of five matches of Go board game [7]. From then on, AI started to invent new algorithms to train the players of different board games.

The main reason behind this development is the invention of Monte Carlo Tree Search (MCTS) algorithm, which could learn and train by itself. In addition to MCTS, Minimax and Alpha-beta pruning technique are also considered as efficient algorithms that could be suitable to defeat the humans [6,14].

Rest of the team members, Zhuoran Yang and Zichen Hao are undergoing their research in studying about the game search and existing AI algorithms related to Chinese chess that could contribute the project to lead in a successful path during the second phase.

1.2. Goal of thesis paper

The goal of this paper is to get well versed in the existing leading AI algorithms that could help in defeating the humans in any board game matches. 50% of success during the second phase of the project depends on this algorithmic study.

1.3. Structure of thesis paper

As per the goal, this thesis is framed to explain MCTS, Minimax algorithm and Alpha-Beta pruning technique in each section (3,4,5) and provides a result of the analysis. The final section describes the future work of the second phase of the project.

2. Literature review:

In general, the AI program was trained in a relatively small domain like games, because only with simulated games, it is possible to generate and analyse extensive data which is impracticable in testing with real life [7]. Since the 1960s, digital computer researchers consider chess as the drosophila of Artificial Intelligence (AI) [16]. From then on, people started to show interest on challenging games which competes with their mind ability [3]. Search and evaluation program of a particular algorithm determines the efficiency of the game [1]. With the combination of selective search and different evaluation functions, new variants in games are performed [3]. This progress in AI evolves other streams like neural network, machine learning, computational intelligence, agent-based reasoning, heuristic search and knowledge representation. To develop intelligent agents with the capability of learning the games by themselves through uploading the rules of games, AI must be tested with different algorithms and board games [1]. Deep Blue was an incredible milestone of AI. This invention won a game against the world chess champion in 1996 after losing three matches and drawing twice. A year later, Deep Blue won by 1 point after inheriting minimax, alpha-beta pruning and evaluation functions. So, it is significant to learn and optimally use the top existing algorithms to defeat humans by AI in board games.

2.1. Minimax Algorithm

The minimax algorithm is also called as negamax algorithm that remains the most widely used search technique being best adaptable for two-player perfect-information games [6]. The MINIMAX algorithm provides an optimal strategy for two-player, even by understanding and tackling the opponent's optimal strategy [12]. Game tree pathology is the key feature of minimaxing, in which the accuracy of the search decreases with the increase in tree height [6]. Minimax algorithms are framed to work in pure trees. So, they work by merging branches and duplicate their work for each merging branch. They are extended with the transition table to avoid duplicating work when branches merge [15]. Using the static evaluation function such as heuristic, computers play a turn-based game by looking at the actions available in the move list that produces a better result than others [15].

Those better moves place the player in a good position which helps them to maximize the score against the opponent who intends to minimize the player's score. So, this change between maximizing and minimizing in the search game tree is known as minimaxing [15]. So, in this game tree the utility values are backed up to the current node (root node), and those are reflected to the optimal strategy’s next move. Minimax has the nature of searching the whole game tree, which requires more effort to reduce the search space [12]. As per the critical principle of Minimax algorithm, scores are bubbled up from the bottom of the tree. E.g., In player A’s turn, the highest score will be bubbled up, and in player B’s turn the lowest score will be bubbled up to player A [15]. In minimax, the scores of the moves are based on one player’s point of view. But it requires a separate code to track whose move it is, to know whether the scores must be minimized or maximized to bubble up [15].

2.2. Alpha beta pruning

In the name of development, more AI algorithms evolved such as alpha beta (AB) pruning technique, negamax, negascout and expectimax algorithm [17]. Among them, random, brute force and greedy algorithm's execution time is fast but return less optimal move than minimax variants. Even though if minimax produces optimal moves, it takes a longer time to execute depending upon the length of its search depth. In this case, it is necessary to choose an algorithm that executes faster with the optimal move at deep search depth. Alpha beta pruning and Memory-enhanced Test Driver (MTD) are the algorithms that satisfy both the standards. However, MTD has certain conditions that limit window size. So, Alpha beta pruning technique is preferred [17]. AB pruning is a mathematically sound technique that helps to detect and prune away the dead branches (irrelevant branches). AB pruning computes similar to the characteristics of minimax, but AB pruning is more efficient because it eliminates the irrelevant subtree. This algorithm can be applied to any tree because it has the capability of pruning entire subtree. AB pruning has a limitation to reach the leaf node of the search tree to take the optimal decision. The common strategy of this algorithm is to cut the search tree by setting a maximum depth. Using heuristic function nodes are evaluated at any depth [7].

2.3. Monte-Carlo tree search:

Among all board games, Go is considered as the most challenging game for AI stream. The Go requires high branching factor which eliminates brute-force-type exhaustive search methods, orders of magnitude with slower static positional analysis and proper positional board judgement requires performing several auxiliary tactical searches oriented in tactical issues [3]. Nicholas Metropolis, the founder of Monte Carlo [Metropolis 1987], this Monte Carlo method, works based on two mathematical theorems: the law of large numbers and the central limit theorem [10]. With the flexibility of the Monte Carlo method it is capable of implementing in varieties of streams like mathematics, statistics, economics, engineering and other quantitative disciplines. The discrete formulation of Monte Carlo is efficient in fuller and continuous framework [10]. This method allows creating a viable answer for complex questions with the help of repetitive histories. (For example histories can lead to algebraic functions [10]. However, the result is still the sum of the histories divided by the number of trials). Based on these principles, this method is used for solving various numerical problems have been used for centuries and saw relatively limited use until the digital computer came into existence [10]. The Monte Carlo method was coined in digital computers to implement statistical sampling. During this implementation, the bits are converted into meaningful rational numbers [10]. Monte Carlo is a flexible and powerful form of quadrature and numerical integration while applying to a wide range of direct and inverse problems. Being flexible Monte Carlo algorithm can solve a given problem in many ways as per the requirement [10]. It is a practical analysis technique to compute long summations with generated pseudorandom numbers. With this ability in the twentieth century, the Monte Carlo formalism was used to estimate the neutrons and other radiation particles [10]. Later, Monte Carlo tree search program is considered as the strong chess program [7]. Similar to Go and chess, for most of the games Monte-Carlo method is found to more appropriate and efficient.

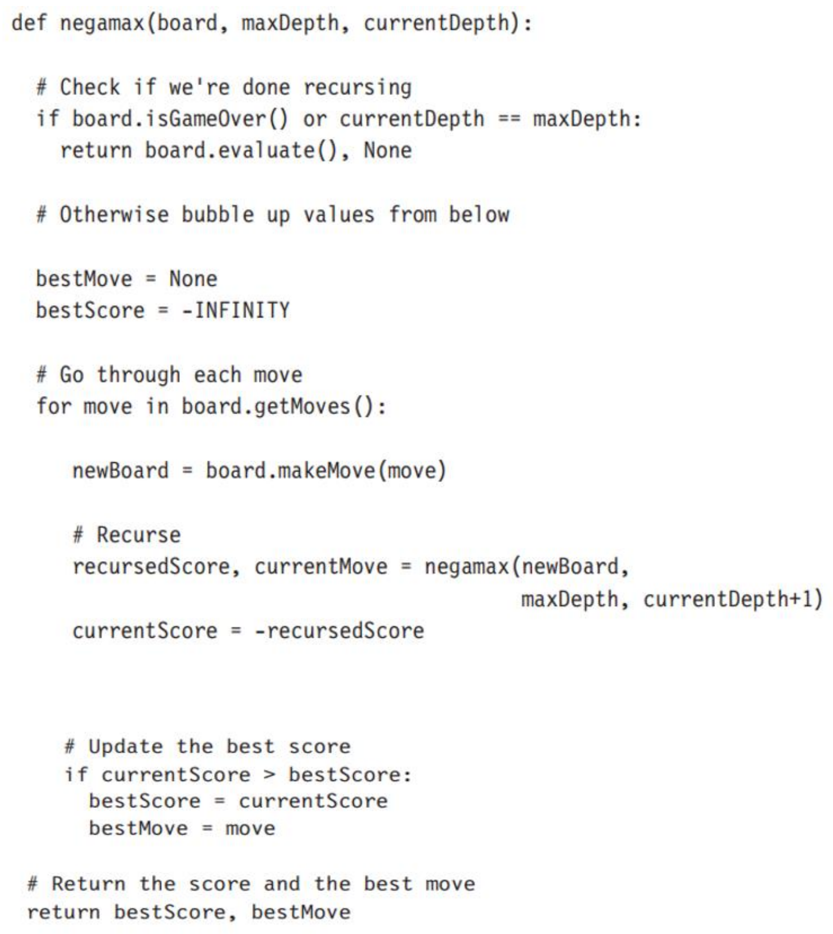
3. Approach

3.1. Minimax Algorithm:

In AI board games, the minimax algorithm conducts recursive. During each iteration, it calculates the correct value of the current player's position in the board and calculates the resulting board position and recurses to get the value of that particular position [15]. The element called maximum search depth is added to the algorithm to stop the search from going on forever, particularly whenever the tree is deep [15]. If the current player's position is at the maximum depth, it calls the static evaluation function and returns the result. If the algorithm considers the position where the current player is to move, then it returns the determined highest value; otherwise, it returns the lowest. This switches the steps between the minimization and maximization. If the search depth is zero, then it also stores the best move found. This will be the next move to make [15].

For example, in chess, if the minimax algorithm is implemented: there are max and min nodes. The white player moves first in chess; it is assumed to get a higher score. Therefore, every node that the white player makes next is a max node because the white player is considered to maximize the score and every node the black player makes corresponds to min nodes so that it could minimize the white's score. Until the value is backed up to the root, this recursive process continues. Once the leaf utility values are defined to the players as per the game requirements, the minimax algorithm recursively operates on the game tree. It starts to iterate between the $\pi 1$ (white) player, which takes the maximum of its children's utility values (on odd-depth states), and the $\pi 2$ (black)player, which takes the minimum of its children's utility values (on even-depth states). After initiating, the utility value is defined with the leaf states of the game tree as a low value for winning states of $\pi 2$ and higher values for winning states of $\pi 1$. Intermediate utility values indicate the advantage of one of the players over its opponent.

In this case, it takes a longer execution time by finding which player point turn. So, to optimise, Negamax is introduced. Negamax alternates the player's viewpoints at each turn and the evaluation function scores from the player point of view depending upon whose turn it is [14]. In order to implement this, the evaluation function could stop accepting a point of view as input and just simply refers to which player's turn it is [14]. The advantages of this method are deterministic strategies are avoided (i.e., eliminating the predictability of the game), the perfect rationality of the player is not assumed, and the control over the strength of the player is allowed.



```
def negamax(board, maxDepth, currentDepth):

  # Check if we're done recursing
  if board.isGameOver() or currentDepth == maxDepth:
    return board.evaluate(), None

  # Otherwise bubble up values from below

  bestMove = None
  bestScore = -INFINITY

  # Go through each move
  for move in board.getMoves():

     newBoard = board.makeMove(move)

     # Recurse
     recursedScore, currentMove = negamax(newBoard,
                                          maxDepth, currentDepth+1)
     currentScore = -recursedScore



     # Update the best score
     if currentScore > bestScore:
       bestScore = currentScore
       bestMove = move

  # Return the score and the best move
  return bestScore, bestMove
```

Figure 1: Negamax Pseudo-Code [15]

This pseudo-code Negamax function can return two things: a best move and its score. For languages that can only return a single item, the move can be passed back through a pointer or by returning a structure. The INFINITY constant should be larger than anything returned by the board.evaluate function. It is used to make sure that there will always be a best move found, no matter how poor it might be.

3.2. Alpha beta pruning

The special feature of AB pruning is of being capable of ignoring the sections of the tree that cannot make best move. Basically, this technique is made up of two kinds of pruning such as alpha (α) and beta (β) [15]. These prunings are helpful to keep track of the best score that could be achieved by the player. α value forms a lower limit on the score that could be attained. This technique might find a better sequence of moves later in the search, but it would never accept a sequence of moves that gives a lower score. This lower bound is called alpha pruning. β value keeps track of upper limit of expected high score and gets updated as it is found [15]. In a situation where there is no way to score more than the beta value and there may be more sequence yet to find that the opponent could stop us. It finds a sequence of moves that scores greater than the beta value [14]. Now, the interval between the alpha and beta value is called as search window. This window considers only new move sequences along with the score and rest are pruned [14]. Instead of calling the algorithm with a range of $(-\infty,+\infty)$, which is called as estimated range [15]. This range is called an aspiration, and the AB algorithm called in this way is sometimes called aspiration search. AB algorithm searches the best move from that estimated range know as aspiration search [15].

Instead of checking the alpha and beta values consecutively, the AB negamax is implemented to swap and invert the alpha and beta values. It then checks and prunes against just the beta value. AB negamaxing is the best method to implement the AB pruning optimally [15]

```
def abNegamax(board, maxDepth, currentDepth, alpha, beta):

  # Check if we're done recursing
  if board.isGameOver() or currentDepth == maxDepth:
    return board.evaluate(player), None

  # Otherwise bubble up values from below

  bestMove = None
  bestScore = -INFINITY

  # Go through each move
  for move in board.getMoves():

     newBoard = board.makeMove(move)

     # Recurse
     recursedScore, currentMove = abNegamax(newBoard,
                                            maxDepth,
                                            currentDepth+1
                                            -beta,
                                            -max(alpha, bestScore))
     currentScore = -recursedScore

     # Update the best score
     if currentScore > bestScore:
       bestScore = currentScore
       bestMove = move

       # If we're outside the bounds, then prune: exit immediately
       if bestScore >= beta:
         return bestScore, bestMove

  return bestScore, bestMove
```

Figure 2: AB Negamax Pseudo-Code [15]

As per Figure 2 Pseudo-code of AB negamax algorithm, the O(d) in memory where d is the maximum depth of the search, and order O(nd) in time, where n is the number of possible moves at each board position. The order of the performance may be the same, but AB negamax will outperform regular negamax in almost all cases. Only if the moves are ordered where no pruning is possible, the AB negamax would be optimal. So, it has extra comparison codes in the algorithm that reduces the execution speed. In majority of cases the performance is very much better than the basic algorithm.

Even though it is efficient in most of the situation, AB pruning technique has drawbacks like the assumption of perfect rationality for both players is unrealistic, it does not address the issues of vast search space for certain games, the behaviour of the player is deterministic and controlling the strength of the player is not feasible [14].

3.3. Monte-Carlo tree search:

Monte Carlo Tree Search (MCTS) is a best-first tree search algorithm. It enables the implementations without evaluation functions. MCTS combined with Upper Confidence bounds applied to Trees (UCT) has an advantage over traditional depth-limited minimax search with alpha-beta pruning in games with high branching factors such as Go [14]. However, minimax search with alpha-beta pruning exceeds MCTS in domains like Chess. Studies shows that MCTS does not detect shallow traps, where opponents can win within a few moves, as well as minimax search. Thus, minimax search performs better than MCTS in games like Chess, which can end spontaneously [14].

The search tree grows larger as the sample size increases because the MCTS samples the promising state and moves instead of fixed-depth minimax search [14]. At the end of every simulation, it reports the rewards as per the randomly played moves till the game gets over [14].

The MCTS method estimates a state of the player by averaging rewards of simulations. This algorithm can be implemented without any domain-based knowledge, but the domain-based knowledge is required to improve the performance [14]

The MCTS implementation follows below two steps:
Step A: Markov Decision Process (MDP): The goal for an MDP problem is to find an optimal policy π which maps states to actions. In other words, a policy specifies what actions should be taken in a given state. Optimal policy means the reward is maximized when decisions are made by optimal policy [14].
Step B: The Monte Carlo method: This method approximates the analytic value by repeated random sampling. By the law of large number, the empirical mean approximates the expected value as the number rises [14]. Therefore, a reliable estimate can be generated by the Monte Carlo method [14].

Below all the phases of MCTS workings are explained [14]:

- Selection Phase: In this phase, UCT is applied. Here, for max nodes, the child node with the highest upper confidence bound (UCB) is selected, and for min nodes, the child node with the lowest confidence bound (LCU) is selected. Now, the algorithm keeps selecting the children nodes until it reaches the last node (terminate node) that stops the process or it reaches nodes whose children nodes are unexplored. If the unexplored nodes are reached, the algorithm enters the expansion phase. If terminated nodes are reached, the algorithm enters the backpropagation phase, which delivers the score of the terminated node.
- Expansion Phase: Here, all the children nodes are expanded. Those expanded nodes are added to the search tree, and they get simulated.
- Simulation Phase: The scores will be propagated until all of its children nodes gets expanded
- Backpropagation Phase: The result of simulations gets returned to parent nodes recursively from leaf nodes to the root node. While returning, the nodes in that path are simulated for one more time and the score increase by one.
- Decision Phase: The children node of root nodes with the highest mean is chosen as the final decision.

MCTS with minimax hybrid algorithm has been proposed to improve the performance of MCTS when shallow traps exist. In this case, the minimax can be embedded into all the phases of MCTS as follows [14]:

- Simulation Phase: A fixed-depth minimax search is done before every random move. Since no evaluation is done, the minimax can only detect proven wins or losses. Thus, the random simulation will find forced wins or avoid forced losses.
- Selection and Expansion phases: A shallow-depth full-width minimax search is done. To improves the MCTS by checking immediate descendants of a subset of tree nodes.

- Backpropagation phase: MCTS backpropagates simulation results to parents.

The MCTS and minimax combined MCTS has a similar framework. It has four repetitive phases and a final decision. Only the methods that algorithms select nodes and backpropagate values are different. There are two significant terms for the Minimax-combined MCTS. They are the Minimax threshold and Minimax node.

- Minimax threshold: The minimax threshold indicates whether a node is included in the minimax search. If the node is visited enough times, then it is referred as a node that meets the minimax threshold [14]. This can be attained using the below formula.

*minimax threshold = minimax threshold parameter * branching factor.*

For example, a node has 3 children and the minimax threshold parameter =50. It meets the minimax threshold if it is visited 150 times.

- Minimax node: The best leaf node which can be reached by the minimax search on the mean from a certain node. If the node does not meet the minimax threshold, the minimax node is itself the best leaf node [14].

4. Result

As per this paper, by analyzing Minimaxing, AB pruning technique and MCTS algorithm each has different usage and drawbacks relative to the board game. But while optimizing them, MCTS combined with minimax provides the best result. While combining the MCTS with minimax the shallow traps are being easy to identify, the depth of search is being controlled, and the execution speed is comparatively higher. So, this paper states that MCTS with minimaxing provides an optimal solution to defeat humans in board games.

5. Discussion

With the help of studying the Minimax, AB pruning and MCTS algorithm, I came to know about the efficiency of all those algorithms and their structure. According to that, only after considering the optimization changes to these algorithms, they are becoming more efficient. Optimizations like Negamax for Minimaxing, AB negamax for AB pruning and combination with minimax for MCTS are undergone. Among those MCTS with minimax are more effective because it gains below three characteristics:

1. Heuristic: Even with full-depth minimax tree search, it does not require any domain-based knowledge. It is easy computational. This has a feature of fixing the depth of search in addition to the optimal evaluation function [14].
2. Anytime: The search tree is built as it increments MCTS. So, it is easy to propagate the results soon after the simulation. This describes the execution speed [14].
3. Asymmetric: The selection policy followed in this method allows MCTS to search on more promising nodes. With this asymmetric shape of the tree, MCTS is able to identify the shallow traps till level-3 of the search [14].

6. Conclusion

In this paper, I have explained about the functions and approaches of three leading AI algorithms for a board game to defeat humans. After the analysis of the characteristics and methodology of these algorithms, I nominate Monte Carlo algorithm as the best suitable algorithm. In future work, I am going to implement Monte Carlo algorithm in a new or existing board game to learn more about the working of this algorithm and functionalities of the board game's software application.

7. References

[1] Shi-Jim, Y. and Y. Jung-Kuei, *Two-Stage Monte Carlo Tree Search for Connect6.* IEEE Transactions on Computational Intelligence and AI in Games, 2011. **3**(2): p. 100-118.

[2] Garcia Diez, S., J. Laforge, and M. Saerens, *Rminimax: An Optimally Randomized MINIMAX Algorithm.* IEEE Transactions on Cybernetics, 2013. **43**(1): p. 385-393.

[3] Mandziuk, J., *Some thoughts on using Computational Intelligence methods in classical mind board games*. 2008, IEEE. p. 4002-4008.

[4] Bjornsson, Y. and H. Finnsson, *CadiaPlayer: A Simulation-Based General Game Player.* IEEE Transactions on Computational Intelligence and AI in Games, 2009. **1**(1): p. 4-15.

[5] Beaudry, E., et al., *Using Markov decision theory to provide a fair challenge in a roll-and-move board game*. 2010, IEEE. p. 1-8.

[6] Abdelbar, A.M., *Alpha-Beta Pruning and Althöfer's Pathology-Free Negamax Algorithm.* Algorithms, 2012. **5**(4): p. 521-528.

[7] Bratko, I., *AlphaZero--What's Missing?* Informatica (Ljubljana), 2018. **42**(1): p. 7.

[8] Campbell, M., A.J. Hoane, and F.-h. Hsu, *Deep Blue.* Artificial intelligence, 2002. **134**(1-2): p.5783.

[9] Chen, J.-C., et al., *Compressing Chinese Dark Chess Endgame Databases by Deep Learning.* IEEE transactions on games, 2018. **10**(4): p. 413-422.

[10] Dunn, W.L., *Exploring Monte Carlo methods*, ed. J.K. Shultis. 2012, Amsterdam ;: Elsevier.

[11] Enzenberger, M., et al., *Fuego-An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search.* IEEE Transactions on Computational Intelligence and AI in Games, 2010. **2**(4): p. 259-270.

[12] Garcia Diez, S., J. Laforge, and M. Saerens, *Rminimax: An Optimally Randomized MINIMAX Algorithm.* IEEE Transactions on Cybernetics, 2013. **43**(1): p. 385-393.

[13] Jr-Chang, C., et al., *Equivalence Classes in Chinese Dark Chess Endgames.* IEEE Transactions on Computational Intelligence and AI in Games, 2015. **7**(2): p. 109-122.

[14] Lin, J.F., *Monte Carlo Tree Search and Minimax Combination – Application of Solving Problems in the Game of Go*. 2017, ProQuest Dissertations Publishing.

[15] Millington, I., *Artificial intelligence for games*. 2nd ed. ed, ed. J.D. Funge. 2009, Boca Raton, Florida ;: CRC Press.

[16] Nathan, E., *Is chess the drosophila of artificial intelligence? A social history of an algorithm.* Social studies of science, 2012. **42**(1): p. 5-30.

[17] Tommy, L., M. Hardjianto, and N. Agani, *The Analysis of Alpha Beta Pruning and MTD(f) Algorithm to Determine the Best Algorithm to be Implemented at Connect Four Prototype*. 2017. p. 12044.

Paper available on request:

[18] Zhuoran Yang, *A study of game search algorithms in Chinese chess*, 2020.

[19] Zichen Hao, *The study of finding an applicable AI algorithm for Chinese chess*, 2020.