Alpha-Beta Pruning in Mini-Max Algorithm –An Optimized Approach for a Connect-4 Game

IRJET Journal

International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 04 | Apr-2018 www.irjet.net p-ISSN: 2395-0072

Alpha-Beta Pruning in Mini-Max Algorithm –An Optimized Approach for a Connect-4 Game

Rijul Nasa[1], Rishabh Didwania[2], Shubhranil Maji[3], Vipul Kumar[4]

[1,2,3,4] *Dept. of Computer Science & Engineering, The National Institute of Engineering, Mysuru, Karnataka, India*

Abstract – *More than six decades after the term "Artificial Intelligence" was coined by John McCarthy to describe intelligent behavior displayed by machines, finally the technology enabled world with automation in every aspect of human life is becoming a reality. Artificial intelligence (AI) is used to describe the intelligence displayed by machines, which defers from the intelligence displayed by humans and other animals often referred to as Natural Intelligence (NI). Right from the inception of AI, game playing has been an area of research. One of the first examples of AI being employed in game playing was the computer game of Nim which was developed in 1951 and published in 1952. After that AI has been used in many computer games like Chess, Tic-tac- toe, and Connect-4. The game of Connect-4 was first solved by James Dow Allen in 1988. Connect-4 Gaming Portal using Adversarial Search (Artificial Intelligence), provides an online interface to the zero sum, perfect information strategy game of Connect 4. A user will be able to log in to the system where he / she can challenge various levels of AI with increasing degree of difficulty. The moves of the AI will be optimized by using mini-max algorithm and alpha beta pruning.*

***Key Words*: Adversarial Search, Zero Sum, Perfect Information, Mini-max Algorithm, Alpha Beta Pruning**

1. INTRODUCTION

Connect-4 is a two-player game in which the players first choose a particular disc color and then take turns to drop their respective colored discs from the top into a seven-column and six-row grid which is vertically suspended. The first player to form a horizontal, vertical, or diagonal line of four of one's own discs is declared winner of that particular game. It is a solved game i.e. a player can always win the game by playing the optimal moves. Over the time many technologies like data mining, computational intelligence etc. have been employed in the game of Connect-4 to make it more efficient and interactive [1]. AI has been used in computer games right from their beginning. A well-crafted, intelligent AI program can make for a challenging—or even impossible to beat—competitor [2]. The focus is on the traditional use of AI by simulating the general game play of the classic board game of Connect-4. Connect-4 is classed to the category of adversarial, zero-sum game because in this game advantage for a player is disadvantage for its opponent.

Under the context of AI, the concept of adversarial search is explored specifically. Here it will examine the problems that arise when someone tries to plan ahead in a world where other agents are playing against it [1]. The games are expressed as multi-agent environments.

In the online Connect-4 gaming interface a user can compete against bots having various levels of difficulty. The various degrees of AI that user can play against will be primarily of nine types-Random AI (This player is full dummy player. It's propose is just to add randomness in the game) and eight AI players utilizing Mini-max algorithm (It prioritizes victory the fastest way possible). The aforementioned eight players represent the eight levels of difficulty, each of which denotes the level of depth to which the decision is computed.

The interface provides user an option to either play the game in which game states will be generated using mini-max algorithm or using mini-max with alpha beta pruning. Section 2 discusses about Connect-4 using mini-max algorithm, section 3 discusses about Connect-4 using min-max with alpha beta pruning, section 4 compares the two algorithms in terms of computation time taken and number of iterations performed, section 5 discusses about heuristic function used to calculate the value of the game states. This is followed by conclusion and references.

2. CONNECT-4 USING MINI-MAX ALGORITHM

In the first option a user plays against the computer in which mini-max algorithm is used to generate game states. Mini-max falls in the category of backtracking algorithm. This algorithm has many applications and is used in decision making and game theory to find the optimal move for a player, assuming that opponent also plays optimally [6]. It is widely used in two player games in which players take turns to make their move such as Tic-Tac-Toe, Chess, etc. As obvious there are two players in mini-max which are called maximizer and minimizer. The maximizer aims to get the highest score possible while the minimizer does the exact opposite and tries to get the lowest score possible. There is a value associated with every board state. For a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. Heuristic function is used to calculate the values of the board. This heuristic function is unique for every type of game.

Consider the perfect binary tree in Fig-1. It has four final states and these four leaves can be reached from the root through various paths. The first player to make the

move is at the root. Considering that the maximizing (or minimizing) player makes the first move, which means it is at root, and opponent at next level. Interesting thing to observe here is that which move maximizing (or minimizing) player makes considering that opponent also plays optimally.

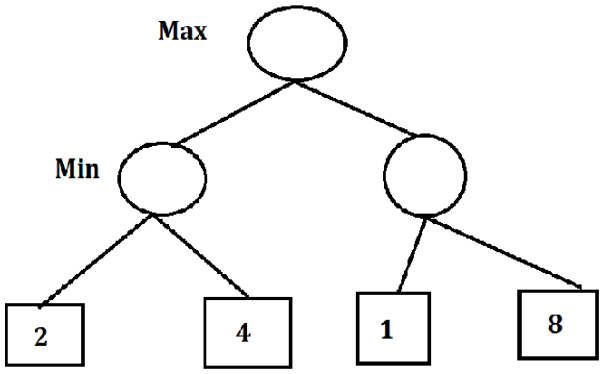

Fig-1: Initial Game Tree in case of Mini-Max

It belongs to the category of backtracking algorithm therefore the decision is made by backtracking after trying all the possible moves. Initially maximizer has the option to go left or right. Assuming it goes left after which it is the minimizer's turn. The minimizer has a choice to make between 2 and 4. The aim of minimizer is to minimize its value therefore it chooses the least value among both, which is equal to 2. Next maximizer goes right after which it is the minimizer's turn. The minimizer has to choose a value between 1 and 8. It will definitely choose 1 over 8. Now it is maximizer's turn to choose between 2 and 1. Maximizer's aim is to get the maximum value possible therefore it chooses the larger value of the both that is 2. So finally the maximizer gets the optimal value of 2 and its best interest lies in going left. The game tree used in the example is very small and is just used for the purpose of describing the concepts clearly and easily but in reality Connect-4 game results in a game tree with a large number of game states. To find and compute each of the game state is virtually impossible. The depth of the tree may be six with a branching factor of seven. Therefore a large number of game states are computed.

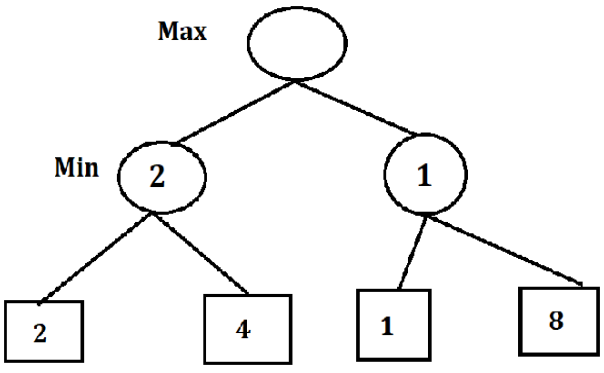

Fig-2: Final Game Tree in case of Mini-max

There are two possible scores based on the left and right moves for the maximizer as can be seen from the above tree. The minimizer will never pick the value 8 from the right sub tree because it is assumed that the opponent always plays optimally.

There are only two choices for a player in the above example but generally, there are many choices. In that case maximum/minimum is found out by recurring all the possible moves. The scores stored by leaves of the game tree for the specified example have been assumed randomly but for a typical game, these values are derived [6].

In the study carried out it is observed that when user plays the game in single player mode and chooses mini-max as the operating algorithm then AI takes 2799 iterations to generate the game state and computation time is 33.00 milliseconds for a difficulty of depth 4.

3. CONNECT-4 USING ALPHA BETA PRUNING

The technique that can be used to optimize mini-max algorithm is the application of alpha-beta pruning. When mini-max with alpha beta pruning is used instead of simple mini-max algorithm then less number of nodes is evaluated in the game tree. Adversarial search algorithm is often used in two-player computer games and this algorithm also falls under this category [6]. Alpha-Beta pruning is not altogether a different algorithm than mini-max; rather it is an optimized version of the mini-max algorithm discussed in the previous section. It optimizes the mini-max algorithm in various aspects like bringing down the computation time by performing less iteration than mini-max. In this algorithm two extra parameters are passed in the mini-max function, namely alpha and beta and that is why it is known as Alpha-Beta pruning. The introduction of two additional parameters make searching much faster and game tree can be searched to much more depth. Game tree has many branches leading to different game states and benefit of using alpha-beta pruning is that if there already exists a better move then other branches need not be searched and can be pruned saving computation time. At any node in the game tree the maximizing player is assured of a maximum score which is stored by alpha and similarly the minimizing player is assured of a minimum value which is stored by beta. These two values namely alpha and beta are updated and maintained by the algorithm. Initially both players begin with worst possible values they can score and therefore alpha is allotted a value of minus infinity and beta is allotted a value of plus infinity. It is possible that when a certain branch of a certain node is chosen the minimum possible score that the minimizing player can get becomes less than the maximum score that the maximizing player is assured of getting i.e. beta is less than or equal to alpha. In this case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, there is no need to explore other branches of that node [7].

Alpha is the best value that can be guaranteed by maximizer at that level or above whereas beta is the best value that can be guaranteed by minimizer currently at that level or above.

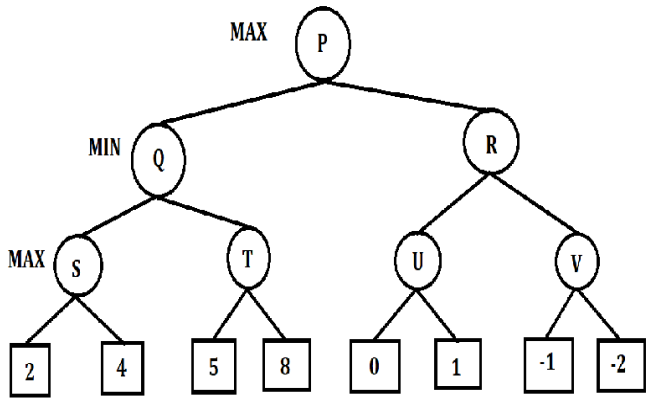

Fig-3: Initial Game Tree in case of Alpha-Beta Pruning

The initial condition of the game tree is shown in Fig-3. Initially the value of alpha is minus infinity and the value of beta is plus infinity. These values are passed down to nodes Q and R in the tree. P is the first node to be called. At P the aim of maximizer is to choose maximum of Q and R, so Q is called first by P but it is not necessary as R can also be called first. At Q the aim of minimizer is to choose minimum of S and T and hence calls S first. At S, left child which is a leaf node returns a value of 2. Now the updated value of alpha at S is maximum of minus infinity and 2 which is 2. To decide whether it should look at its right node or not, it checks the condition beta is less than or equal to alpha. This is condition does not hold true since beta is equal to plus infinity and alpha is equal to 2. So, the game tree is searched on. S now looks at its right child which returns a value of 4. At S, alpha is equal to maximum of 2 and 4 which is 4. Therefore, finally the value of node S is 4. S returns this value of 4 to its parent node Q. At Q, beta is equal to minimum of plus infinity and 4 which is 4. The minimizer is now guaranteed a value of 4 or lesser. Q will now call its right child T to check if it is possible to get a value lower than 4. At T the values of alpha and beta is not minus infinity and plus infinity but instead minus infinity and 4 respectively, because the value of beta was changed at Q and that is passed down by Q to T. Here T checks its left child which is 5. At T, alpha is equal to maximum of minus infinity and 5 which is 5. Here the value of beta is 4 and value of alpha is 5 therefore the condition beta is less than or equal to alpha holds true. Hence T does not need to check for its right child even though it stores a larger value than 5. Here right branch of node T is pruned and T returns the value of 5 to Q. Here no matter what the value of T's right child is, it is not explored. Even if the value is plus infinity or minus infinity, it still wouldn't make any difference. It is not necessary to even look at it because the minimizer is guaranteed a value of 4 or lesser. The basic idea behind this logic is that as soon as the maximizing player got the value of 5 it knew the minimizing player can get a lesser value of 4 on the left side of node Q and if it comes to right side of Q it will get a value of 5 or more which in any case in greater than 4 and hence will never come to the right side and therefore there is no need to waste computation time by exploring T's right child which stores a value of 8. This way this algorithm works faster than simple mini-max algorithm. Now T returns a value of 5 to Q. At Q, beta is equal to minimum of 4 and 5 which is equal to 4. Therefore, node Q stores a value of 4.

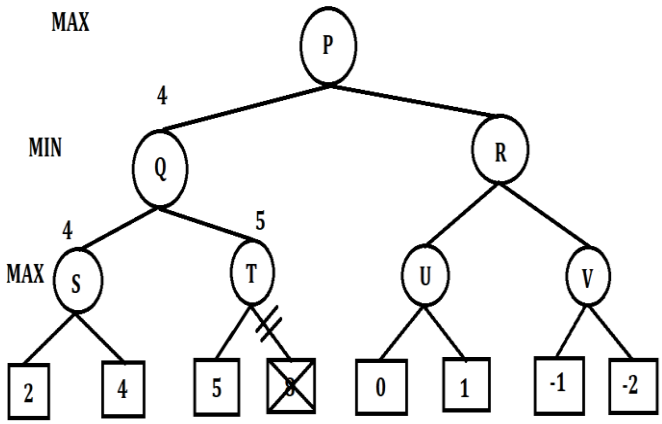

Fig-4: Intermediate Game Tree in case of Alpha-Beta Pruning

So far this is the condition of game tree. The right branch of node T which stores the value of 8 is crossed out because it was never computed. Node Q returns the value of 4 to its parent node P. At node P, alpha is equal to maximum of minus infinity and 4 which is equal to 4. Here the interesting thing to note is that the maximizer is guaranteed a value of 4 or greater. P now calls its right child R to check if it is possible to get a value higher than 4. At R, alpha is equal to 4 and beta is equal to plus infinity. Here R calls its left child U. At node U, alpha is equal to 4 and beta is equal to plus infinity. Node U looks at its left child which stores the value of 0 therefore the updated value of alpha is equal to maximum of 4 and 0 which is still 4. Right child of U stores a value of 1. Hence the best value this node can achieve is 1 but alpha still remains 4. U returns a value of 1 to R. At R, beta is equal to minimum of plus infinity and 1. The condition beta is less than or equal to alpha becomes false as beta is equal to 1 and alpha is equal to 4. So, the further search for deeper levels breaks and there is no need to compute the entire sub-tree of V. The intuition behind pruning of entire sub-tree of V is that, at node R the minimizer is assured a value of 1 or lesser but before that maximizer was already guaranteed a value of 4 if it goes to its left and chooses node Q. So, it does not make any sense for maximizer to go to its right and choose node R and get a value less than 1. Again, it does not matter what values were stored by both children of node V. Hence it can be seen how alpha-beta pruning algorithm saves a lot of computation time by skipping an entire sub tree. Now node R returns a value of 1 to its parent node P. Therefore, the best value at P is maximum of 4 and 1 which is equal to 4. Hence the optimal value that the maximizer can get is 4. The final game tree is shown in Fig-5. The entire sub-tree V has been crossed out as it was never computed.

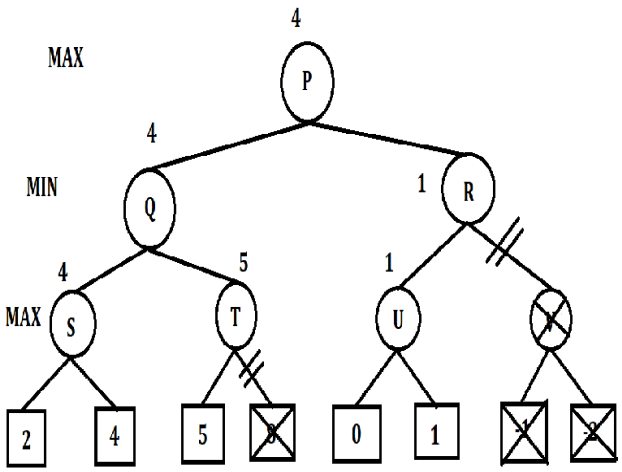

Fig-5: Final Game Tree in case of Alpha-Beta Pruning

In the study carried out it is observed that when user plays the game in single player mode and chooses mini-max with alpha-beta pruning as the operating algorithm then AI takes only 477 iterations to generate the game state and computation time is only 6.00 milliseconds for a difficulty of depth 4.

4. COMPARISION BETWEEN MINI-MAX AND ALPHA-BETA PRUNING

As discussed in previous sections it is quite evident that mini-max and alpha-beta pruning are not entirely different algorithms rather alpha-beta is just an optimized version of simple min-max algorithm.

Algorithm Used	Mini Max		Alpha-Beta Pruning	
Difficulty Level	No. of Iterations performed	Computation Time (ms)	No. of Iterations performed	Computation Time (ms)
Depth 1 (Easiest)	7	0.00	7	0.00
Depth 4 (Normal)	2799	33.00	477	6.00
Depth 8 (Hardest)	5847005	55441.00	71773	1009.00

Table-1: Comparison between Two Algorithms

In the study carried out it is observed that when the user plays the game in single player mode and chooses mini-max as the operating algorithm then AI takes 7 iterations to generate the game state and computation time is 0.00 milliseconds for a difficulty of depth 1 whereas for the same difficulty level alpha-beta pruning also takes 7 iterations and 0.00 milliseconds to compute. The difference between the two algorithms becomes starker when difficulty level is increased as it is clear from the observation that for a difficulty of depth 4 AI takes 2799 iterations to generate the game state and computation time is 33.00 milliseconds in case of mini-max whereas for the same difficulty level alpha-beta pruning takes only 477 iterations and 6.00 milliseconds to compute. Further it is observed that when difficulty level is increased to 8 which is the toughest level AI takes 5847005 iterations to generate the game state and computation time is 55441.00 milliseconds in case of mini-max algorithm whereas for the same difficulty level alpha-beta pruning takes only 71773 iterations and 1009.00 milliseconds to compute. The result has been summarized in table-1.

5. HEURISTIC FUNCTION

The application of heuristic function is to coordinate robot agents in adversarial environment [4]. The AI will exhaustively evaluate the tree to a certain depth (the number of moves to "think ahead") and then evaluation of the board is done using heuristic function. Eventually, as the end of the game draws near, the AI finds winning or losing states confined to the specified depth and plays perfectly. The whole idea here is that AI can be guided to a winnable position by heuristic function. A function is implemented that calculates the value of the board depending on the placement of pieces on the board. This function is often referred to as Evaluation Function and sometimes also called Heuristic Function [3]. The basic idea behind the heuristic function is to allot a high value for a board if maximiser's turn is being played or a low value for the board if minimizer's turn is being played.

In the actual algorithm it is generateComputerDecision()'s responsibility to calculate the most optimal move for the AI to play. It is to be kept in mind that the AI is the maximizer and human is the minimizer. The aforementioned function calls upon the maximizePlay() function with current board and the search depth as parameters. The two other parameters are alpha and beta. Alpha is the best value that the maximizer currently can guarantee at that level or above. Beta is the best value that the minimizer currently can guarantee at that level or above [6]. The first thing that is done is the calculation of the score (value) of the present board. Next check if the current state is a terminal state or not. The next step is to create multiple game states for the next iteration or stage. For each game state created call the minimizer function. The minimizer function is structurally same as the maximizer function. The board score is computed first. Then it is checked if the game state is a terminal state or not. A loop is run to generate the next iteration of game states. For each game state generated the maximizing move for the AI is calculated. The principle of alpha beta pruning is applied to reduce the number of game states that needs to be checked. This function returns the minimum score of the human player.

The heuristic function calculates two parameters, namely human points and computer points. Next, two arrays are initialized whose job is to save the winning position of the respective player. The next step is to determine score

through the amount of available chips. It is done through incrementing the human or computer points whenever the particular field of the game state is empty or not. The final stage is to check if the current state is a winning condition or not.

The actual function that calculates the value of game state calculates five parameters, namely vertical points, horizontal points, diagonal one point, diagonal two points and the final points. Next a nested loop is run for each column, where we rate the column according to the aforementioned function and the points to the respective column. The same procedure is carried out in calculating the horizontal points and the two diagonal points. The final points are nothing but the summation of the former four points.

6. CONCLUSION

In this paper the game of Connect-4 has been implemented using two algorithms and comparison between them is studied. First algorithm is mini-max algorithm and second one is mini-max with alpha beta pruning which is an optimized version of mini-max algorithm. The study revealed that for the same level of difficulty the two algorithms behave very differently in terms of number of iterations performed and time taken with alpha beta pruning taking much less time and performing very few iterations than mini-max to generate the game state.

REFERENCES

[1] Ana Carolina Ribeiro, Luis Miguel Rios and Ricardo Meneses Gomes "Data mining in adversarial search — players movement prediction in connect 4 games" 2017 12th Iberian Conference on Information Systems and Technologies (CISTI).

[2] Samineh Bagheri, Markus Thill, Patrick Koch and Wolfgang Konen "Online Adaptable Learning Rates for the Game Connect-4" IEEE Transactions on Computational Intelligence and AI in Games.

[3] Simon M. Lucas and Thomas Philip Runarsson "On imitating Connect-4 game trajectories using an approximate n-tuple evaluation function" 2015 IEEE Conference on Computational Intelligence and Games (CIG).

[4] I. Levner, A. Kovarsky and Hong Zhang "Heuristic search for coordinating robot agents in adversarial domains" Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.

[5] http://www.geeksforgeeks.org/category/algorithm/game-theory/.

[6] https://en.wikipedia.org/wiki/Alpha-beta_pruning