International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 5 May 2022, pp: 1437-1442 www.ijaem.net ISSN: 2395-5252

Min-Max Algorithm

Raunak Chaurasia, Siddharth, Rohit Gupta, Vivek Yadav

Submitted: 15-05-2022 Revised: 20-05-2022 Accepted: 25-05-2022

ABSTRACT
The modern-day transformation, that is into the digital world from an existing physical world, is brought with the help of Information Technology.When it comes to various fields, especially in terms of games, the transformation has developed and has been vital. With the help of the paper, the aspect of involvement of Artificial intelligence, in games involving physical objects like mahjong, chess, cards and even dominoes, which are quite popular in public. With the help of this paper, we will discuss over Min-Max Algorithm while discussing its applications too. The advantages as well as different ways of improving the algorithm will be discussed along the path too.

Index Terms: Artificial Intelligence, games, min-max, information technology

I. INTRODUCTION

Let's assume, the individual is planning to play a game with his/ her friend. Here, the bar or the level of "performance in the game" is monitored by a numeric value, that can be observed as increasing when, the individual gets closer to "winning the game", compared to the rival, while, it decreases when the rival is closer to "winning the game". It is with this setting,that one can observe the individual, increasing his/her score (by maximising the score) while the rival, trying to decrease the individual's score (by minimising the same).

This situation, calls for an algorithm, which can make decisions(good in nature); leading to winning the game. The modelling takes play in the following manner: two(2) entities i.e. functions will be used, for calling each other, while one works on maximising the score, while the other works totally opposite to it, i.e. minimising. In layman's language the functions will be mimicking the players.

The algorithm is an amazing example of AI, which isn't equivalent to ML. Actually, ML is actually a subcategory of AI. ML isn't involved in some of AI techniques. The algorithm, also referred as " Mini-Max" makes the machine(computer) behave much more intelligently but, with the drawback of not being able to learn anything. Despite such drawback, it turns out to be quite functional in many games.

The algorithm is usually useful in a turn-based game, moreover, involving 2-players. The minimax algorithm helps in decision-making, with the main objective being: to the best possible and optimal move.In the algorithm, one is called maximiser, while the other is referred as minimiser. When an assessed score is assigned to the game board, one player tries to choose a state with maximum score while the other tries completely opposite, i.e. choose the state with minimum score.

The concept used here, is often referred as **"Zero-Sum Concept"**. It is with this concept, where the total utility score isfound to be divided among the players. Since, the concept uses such concept, an increase in score of an individual, affects the score of others, that is decrease in their score. It is due to this phenomena, that the total sum of the score turns out to be zero always. The game ends when one of the player wins while the other loses. Examples of such games are chess, tic-tac-toe, checkers and poker.

II. FORMULATION OF GAME PROBLEM

The game consisting of participation of 2 players, one being MAX, while the other being MIN; given that the MAX playeris the first to take the turn, the search problem can be defined as:

- Initial State: The current board position
- Player: The active user/ individual, who plays the move
- Successor Function: A list or set of all legal pairs, where the pair being (move,state)
- Goal Test: Represents the terminal states, i.e. whether the game has ended or not
- Utility Function: Provides us with the numerical value for the terminal state, i.e. win or draw or lose The game tree is formulated by summing up of details like initial state and legal moves.

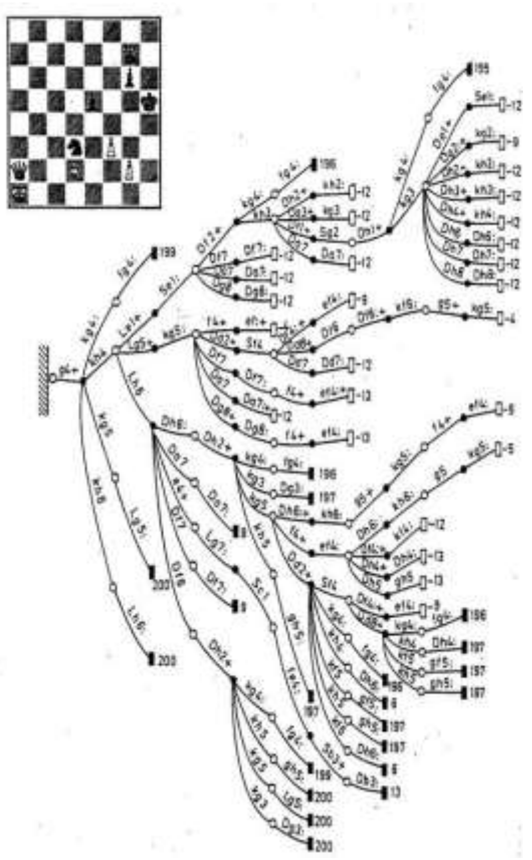

Fig. 1: Example of a game tree for chess

III. DESCRIPTION OF MINIMAX

In simple words, the 2-player game taken into consideration is **Chess**. So, the players i.e. Max and MIN keep on takingturns, one after the other, which can be represented using a tress of decisions. Let’s look at an example:

As per the Fig. 2, the decisions are represented using nodes, except the ones present at the end(terminal nodes). The decision for the move to be chosen, is taken by the individual. It is in this example, that we ought to select from only 2 movements, but, in actual case scenario, we can have any no. of movements whose no. is found to differ from node to node based uponon the state of the present situation.

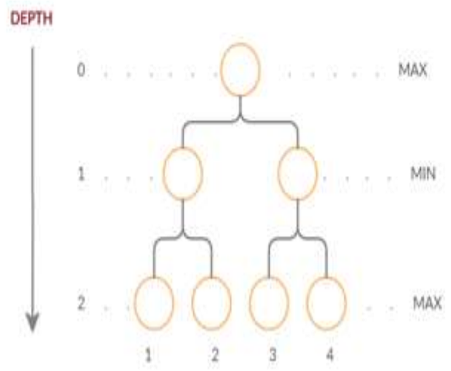

Fig. 2: Simple Example

- Current state is represented by **depth-0** i.e. the top most node. It is the position where, the decision of the next movewill be made. It is the MAX player, who will initialise with what we actually want to do: that is maximising the score. Being a smart player, the MAX player thinks about how the other player(MIN) would respond to the move selected byhim, in place of just selecting out of the possible moves.
- Now, MAX calls MIN, asking him what move would he play if MAX chose either the left or the right one.
- On the basis of moves which MIN states he would play for both(left or right), the MAX will choose the move, which provides him with the maximised score.
- This leads to the question, on how MIN will choose his move, in order to minimise the score. The answer being quite simple, he will ask MAX about the same, i.e. what movements would he choose as a response, if MIN chooses any of the possible choices of moves. Here, the only twist being, the MIN will do completely opposite, that is make a choice which minimises the score.
- This leads into formation of a big tree that has recursion fashion until they reach the last layer of depth i.e. terminalnodes.
- The terminal state is preferably regarded as the state at which, the game will end. Since, the computational expense of exploring all the possible moves, until the end of the game is found- is very high with the processing time being too long. So, to avoid the same, the maximum depth is set. The state would be considered terminal, when the game is either overor the depth limit (maximum depth level) is reached.
- The terminal states for Fig. 2 are the ones

present on the bottom i.e. at depth level 2.

So, the question arises, what happens at terminal nodes. The player irrespective of whosoever it is that is MIN or MAX, won't be able to call the other player i.e. use the same strategy. It is with the help of these terminal nodes, that calculation of score for the game is carried out. Estimating the numeric value of score based on the current state of the game might not be an obvious choice, but, is a necessity in the algorithm.
In the example that we have taken, the terminal nodes are the ones, which are numbered as 1,2,3,4. The only possible option that can be opted by the Max player, is to return these scores over to the previous level, where the Min was calling.
It is through this process, that we will get to the following position:

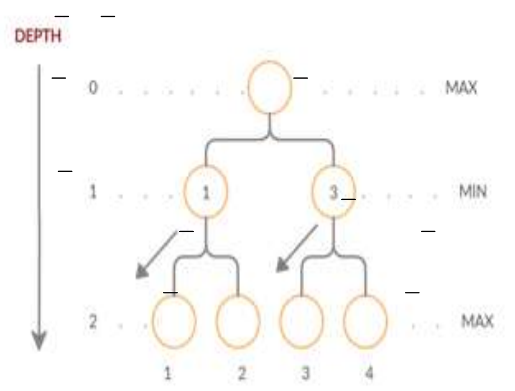

Fig. 3: Position-1

The options which are chosen by MIN are marked with the help of an arrow. This leads to propagation_ of score i.e. numeric values from depth level-2 to depth level-1(MIN's level).
After obtaining the score on_depth level-1, MAX will be able to perform his move, i.e. on level-0: As shown in Fig. 4,

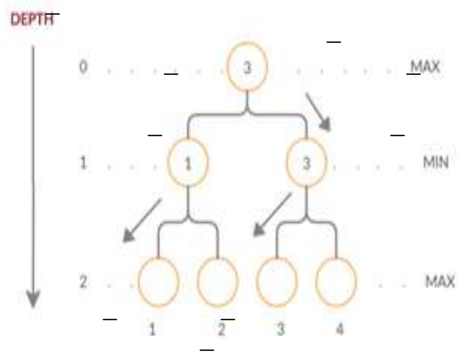

Fig. 4: Position-2

The most optimal move that can be carried out, is the one represented by right edge of the respective top-node, resulting in score of numeric value of atleast 5(originated from node 1 at terminal node).
The word "atleast" is used as the minimax-algorithm runs, stating that both the players(MIN and MAX) are choosing the best move. But if the opponent turns out to be worse than MIN, a better score will be achieved.

IV. IMPLEMENTATION

The tree provided or generated above, is just a mere example to describe it intuitively. The computer isn't required to construct such tree explicitly. The functions which are needed in this algorithm are: min(to minimise the score) and max(to maximise the score).

```
function max( state ):
if it is terminal( state ):
return (NULL, evaluate( state ))
( max state , maximum score ) = (NULL, −infinite )
for child in state . children ( ):( , score ) = min( child )
if score > maximum score :
( max state , maximum score ) = ( child , score )
return ( max state , maximum score )
```

the **max()** is responsible for returning a tuple, that consists of child state at the first position while, the other position is the estimated score which will be achieved by choosing that state. The child state is responsible for maximising the score here. the **min()** is a function returning complete analogue quantity to the **max()** function.

```
function min( state ):
if it is terminal( state ):
return (NULL, evaluate( state ))
( min state , minimum score ) = (NULL, infinite )
for child in state . children ( ):( , score ) = max( child )
if score < minimum score :
( min state , minimum score ) = ( child , score )
return ( min state , minimum score )
```

Now, we will be adding a **decision()** function, that takes the current state as an input and provides the state that will give us the maximised score in return.

```
function min( state ):
( max state , ) = max( state )
return max state
```

The **evaluate()** is a function, that will assist in predicting the score of the state provided as an input.

V. PROPERTIES OF ALGORITHM

- **Complete**: It is ought to find the solution(if it exists), from the finite search tree.
- **Optimal**: It turns out to be optimal, if the choices of both players turn out to optimised.
- **Time Complexity**: Since it is observed to follow Depth- First Search for traversing through game tree, complexity is found to be **O(b^{m})**, where ”b” is the branching factor and the maximum depth of tree is defined by ”m”.
- **Space Complexity**: It is similar to the Depth-First Search ,i.e. **O(bm)**

VI. IMPROVEMENT

For large number of problems, it is found that construction of an entire tree based upon decisions isn’t feasible. So, in practice, a limited(in terms of depth) tree is developed. Also, the evaluate() function is used, to determine the level of goodness of current state, for the player.

As pointed out before, the computational time for the limited game tree is very high, due to high branching factor.

So, fortunately, without exploring each and every node, there’s still a way of finding the optimal move. It is by following some basic rules, skipping of some of branches could be allowed, that won’t have any affect over the final output/result. This phenomena is referred to as **pruning**. A variant for the minimax- algorithm is **Alpha-beta pruning**. Let’s take the following example:

So, now the question which comes to the mind, is, whether we need to examine all the nodes present within the tree or not. It is with the help of this improvisation, i.e. Alpha-Pruning strategy that ignores some branches of the tree which are known to play no major role in making the optimal decision, prior to the time of exploring. As the strategy states, 2 parameters are used in the algorithm, i.e α and β.

It is in this method, that we propagate the established score of the node in upward direction, i.e. to its parent node in the tree. We utilise the score to set the upper or lower bound of the result at the parent node.

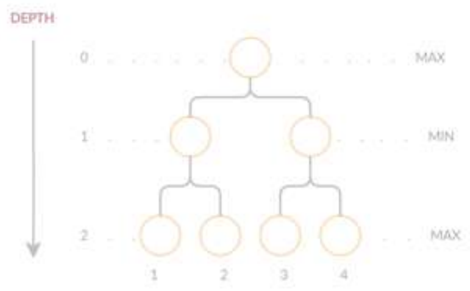

Fig. 5: Example Considered

VII. ANALYSIS WITH EXAMPLE

For instance, when we take our tree, while evaluating the nodes (i.e. from left to right), we are able to establish that the numerical value or the score of terminal node(i.e. leftmost, here value being 4), we are able to state that the result from the parent node, can’t be greater than 4. It is because of the fact that the parent node is a ”MIN” due to which reason, if the value is 4, the value as an output couldn’t be chosen as something much larger.

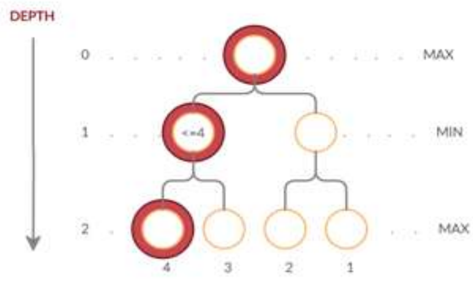

Fig. 6: Position considered

But actually, it is a bit early, i.e. to state that the information is of much help. Now, evaluation of the other node is carried out. Now, we find that the value associated with the parental node is equal to 3. Due to this information, now, we can state that the top node cannot be smaller than 3, turning out to be an important fact.

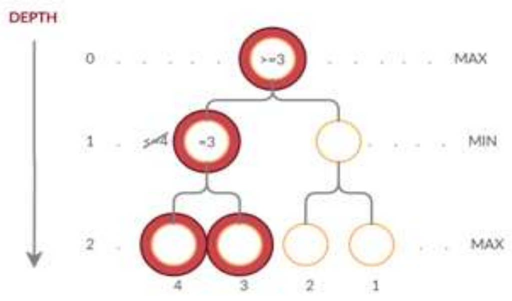

Fig. 7: Position after visiting another node

On continuing the evaluation, we observe one of the score of terminal node turned out to be 2, stating that the value forparent being <=2. Now the question arises, whether we need to evaluate the last node or not. Since, from the previous nodes, it was clear that the top most node could either have 3 as its least possible score, but, from the right branch, it is found that the value cannot be more than 2. So, due to the same reason, we will ignore the right most branch at depth level-1.

The small example, where only one terminal node was skipped, might not seem to have a huge impact i.e. a great improvement. But, such branch in a larger tree, might hold a sub-tree which might be larger in terms of depth. In our

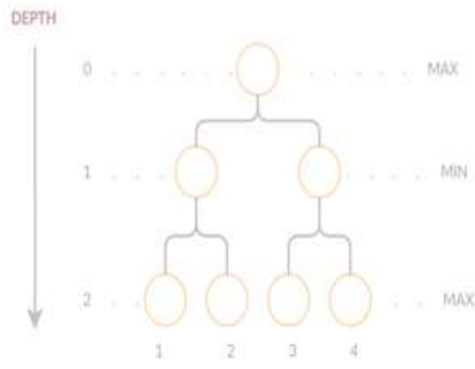

Fig. 8: Position after continuing evaluation

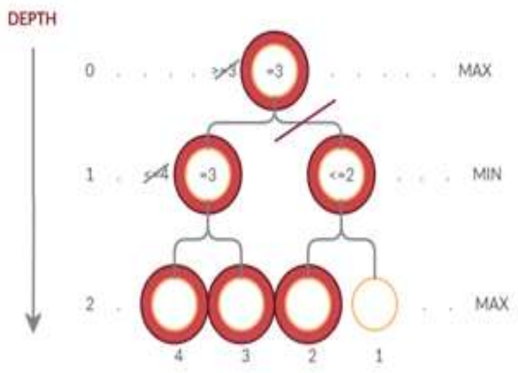

Fig. 9: Position after evaluating right branch

case, we were able to skip only one of the sibling node out of the right-most branch, but there's quite a possibility that, thatnode turns out to be a bunch of nodes, which would've been skipped too. For instance, if the branch(right) had 2,1,0 as the terminal nodes,after travelling through node with 2 as its score value, we could have skipped 1 and 0.

It is on the basis of the ordering of the terminal nodes, that the no. of nodes that will be skipped, will be determined. Itis due to the same reason, in α - β pruning, scores were reversed to 4,3,2,1 from 1,2,3,4 i.e. not assigned randomly. When ascending order is observed, worst case scenario is found, where no pruning could be performed. But with the help of α - β pruning, that the minimax algorithm is able to parse twice as deep when compared with no pruning for the same period oftime.

VIII. PSEUDO-CODE FOR IMPROVISED TECHNIQUE

```
function max( state ):
if it _is _terminal( state , alpha , beta ):
return (NULL, evaluate( state ))
( max _state , maximum score ) = (NULL, −infinite )
for child in state.children():
( _, score ) = min( child , alpha , beta )
if score > maximum score :_
( max state , maximum score ) = _( child , score )
if maximum score _>= beta :
break
if maximum score _> alpha :alpha = maximum score
return ( max _state , maximum score )
```

Alpha referred in the pruning technique, is the biggest lower bound amongst the MAX parent nodes while the Beta is the smallest upper bound amongst the MIN parent nodes. While considering Max function, through the iterations over child nodes, updating of alpha value to the max. score is carried out. If the maximum score is found to be greater than beta, it indicatesthat there exists a parent Min node, which won't choose the current branch, thereby exploring the remaining child nodes. The analog is carried out in min function.

```
function min( state ):
if it _is _terminal( state , alpha , beta ):
return (NULL, evaluate( state ))
( max _state , maximum _score ) = (NULL, infinite )

for child in state . children ():
( _, score ) = max( child , alpha , beta )
if score < minimum score :_
( min state , minimum score ) = _( child , score )
if minimum score_<= alpha :
break
if minimum score_< beta :beta = minimum score
return ( min _state , minimum _score )
```

The +/- infinite is used within the decision function(first call being max()), to initialise the algorithm, thereby taking care ofthe factor, that the output score isn't restricted.

```
function min( state ):
( max state_, ) = max( state , −infinite , infinite )
return max state_
```

IX. CONCLUSION

Minimax Algorithm turns out to be one of the most used/popular algorithms when it comes to computer board games. The applications are focused more on turn-based games. If information regarding the game is known to the players, the algorithm turns out to be the best choice.

However, if the branching factor is found to be very high(for eg. GO), the algorithm doesn't turn out to be quite that beneficial as the time required for processing is quite high. But, with a given proper implementation, the AI can turn out tobe pretty smart.

ACKNOWLEDGMENT

We would like to appreciate all those who provided me the possibility to complete this term paper. We acknowledge with thanks, the support and guidance rendered by our faculties in the A.I. field. We also appreciate the support and suggestiongiven by my friends.
We are extremely grateful to our family members for the support and encouragement to help us complete this term paper.

REFERENCES

[1] https://arxiv.org/abs/math/0501315
[2] https://arxiv.org/abs/math/0609825
[3] https://arxiv.org/abs/0705.2404
[4] https://towardsdatascience.com/understanding-the-minimax-algorithm-726582e4f2c6
[5] https://www.cpp.edu/ ftang/courses/CS420/notes/adversarialsearch.pdf
[6] https://www.javatpoint.com/mini-max-algorithm-in-ai
[7] https://www.baeldung.com/java-minimax-algorithm