Int. j. adv. multidisc. res. stud. 2023; 3(3):463-467

Received: 10-04-2023
Accepted: 20-05-2023

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

Algorithms Searching for Solutions in Fighting Games

[1] Nguyễn Thu Hương, [2] Nguyễn Phương Vân
[1, 2] Thai Nguyen University of Technology, Thai Nguyen University Thai Nguyen, Vietnam

Corresponding Author: **Nguyễn Thu Hương**

Abstract
In this Paper, we introduce Algorithms searching for solutions in fighting games, a common layer of problems in the field of Artificial Intelligence (AI). There are many types of fighting games. The complexity of algorithms in terms of memorial space and computational time is often very large. Each particular game has its own rules of movements, starting states and ending states. In order to describe this problem, 5 components including starting state, intermediate state, ending state, transition, and search tree are used. Finding a new move advantageously is a search tree basing on the evaluating function. From the general algorithm, in order to reduce the complexity, the paper also introduces the α-β cut method. According to this method, some branches in the search tree can be ignored. When applying the limited search depth level and the α-β cut method, the algorithm produces a faster searching result and allows the player to have multiple choices about the difficulty level of the games.

Keywords: Artificial Intelligence, Minimax, α-β Cut

1. Introduction

Artificial intelligence or artificial intelligence (AI) is an intelligence programmed by humans with the goal of enabling computers to automate intelligent behaviors like humans [1, 2]. The difference between artificial intelligence and logical programming in programming languages is the application of machine learning systems simulating human intelligence in processes that humans do better than computers [3].

Artificial intelligence has been known as a keyword of the 4th industrial revolution. The world is entering a new era with rapid development of thanks to AI. AI is the realization of human abilities such as seeing, hearing, decision-making, movement and learning entered into computers. In other words, AI is a combination of algorithms, big data, and computing power. These are the three main components that allow us to build the complete artificial intelligence [4-6].

Artificial intelligence has become a bridge for many fields from digital, physical to biotechnology. Devices and machines using AI can be present everywhere in life [7-10]. Amazon, for example, distributes billions of products using advanced automation technology, machine learning and robotics, maximizing efficiency, minimizing human resources [12, 13].

In the core of this paper, the algorithm searching for the solution in the game with 2 opponents (such as chess, Chinese chess, checkers, etc.) is proposed [11]. The first player is called White, his opponent is called Black. The aim of the algorithm is to search for White to get the best benefit for itself and pass the difficulty to the Black (the algorithm is set up on the side of White). The algorithm will stop when neither White win the game or Black cannot move to the next step.

2. Search Space

We will consider two-player games with the following characteristics. Two players take turns making moves that follow certain rules, the rules are the same for both players. Typically, in chess, two players can apply the rules of pawn, castle, to make a move. Rule of white pawn, white castle, etc. as well as the rule of Black pawn, Black castle, etc. Another feature is that both players are fully informed about the situation in the game (unlike in card games, the player cannot know what cards the other players have left). The chess problem can be thought of as a problem of finding a move, where at each turn, the player must choose amongst many possible options (obeys the rule), the best option which could lead him to win in the player’s turn of games. However, the search problem here will be complicated because of opponents, the player does not know the movements of his opponent in the next steps. We will state more precisely this search problem in the following:

- The chess problem can be considered as the search problem in a state space. Each state is a situation (the arrangement of the chess pieces of the two sides on the chessboard).
- The starting state is the arrangement of the chess pieces of the two sides at the beginning of the game.
- Operators are valid moves.

- Ending states are situations where the game stops, usually determined by some stopping condition.
- A play-off function that corresponds to each ending states with a certain value. For example, in chess, each ending state can only be a win, a loss (for White) or a tie. Therefore, we can define the play-off function as one that takes the value 1 at ending states that are winning (for White), -1 at ending states that are losing (for White), and 0 at ending states that are losing (for White). In some other games, such as a scoring game, the pay-off function takes an integer value in the interval [-k, k] where k is some positive integer.

So, the problem of White is to find a sequence of moves alternating with Black's moves producing strategy from starting state to ending state which White wins.

2.1 Search Tree

To facilitate the study of move selection strategies, we represent the above state space in the form of a game tree.

The game tree diagram is constructed as follows. The root of the tree diagram corresponds to the starting state. We call a node corresponding to the state at which White (Black) decides its option for movement White node (Black node). If one node is White (Black) corresponding to a state u, the nodes following include all nodes representing a state v, where v obtained from u when White

(Black) has performed a valid movement. Therefore, on the same tree level, all nodes are White (Black), the leaves correspond to the ending states.

Consider the game Dodgem (created by Colin Vout). There are two White chess pieces and two Black chess pieces initially placed on a 3*3 board (Fig 1). Black could move to the blank box either on the right, above or below. White could move a blank box either on the left, on the right, or above. Black when it is on the right, most column could move out of the board, White when it is on the top row could move out of the board who either can move out of the board first, or push the opponent cannot move will win. Whoever takes his two pieces off the board first wins, or creates a situation where the opponent can't move will also win.

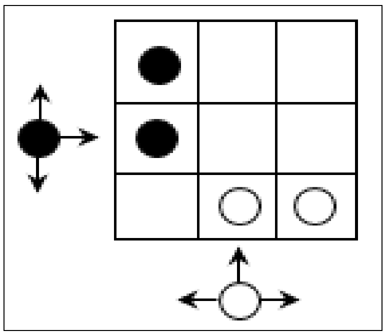


Fig 1: Game Doegem

In the situation Black goes first, the game tree shown in Fig 2.

When Black moves first, each branch of the tree diagram represents a selection of Black chess pieces to expand state space of the game.

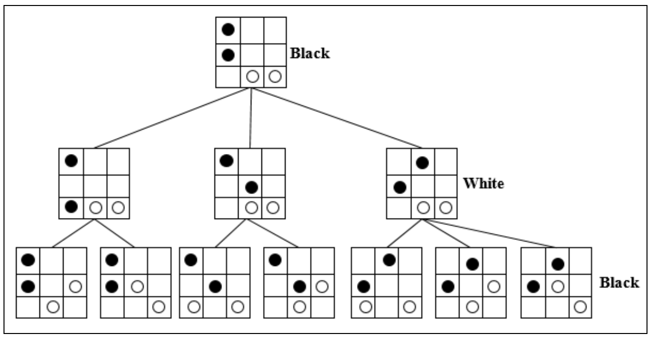


Fig 2: Search tree

2.2 Searching on the Game Tree

Chess is a process in which Black and White decide alternatively and one of the valid moves. On the game tree, that process creates a path from the root to the leaves. Suppose at some point, the path has led to the node u. If u is a White (Black) node, White (Black) needs to have decision to move to one of the Black (White) node v that is a sub-node of u. At the Black (White) note v that White (Black) has just selected, Black (White) needs to move to one of the White (Black) nodes w which is a sub-node of v. The above process will stop when reaching a node which is the leaf of the tree diagram.

3. Minimax Strategy

In the situation White needs to find a move at u node. The optimal move for White is the one that gradually reaches the sub-node of v which is the best node (for White) among the sub-nodes of u. We assume that, in opponent turn moving from v, Black also choose the best move. Thus, to choose the optimal move for White at node u, we need to determine the values of the nodes of the game tree diagram with root u. The value of the leaf nodes (corresponding to the ending states) is the value of the play-off function. The larger the value of the node, the better for White and vice versa. To determine the value of the nodes of the game tree with root u, we move from the lowest level to the root u. Assuming that v is the inner node of the tree and its sub-nodes have been determined and if v is a White node, its value is determined to be the largest in the values of the sub-nodes. If v is a Black node, its value is the smallest in the values of the sub-nodes.

The Fig 3 shows the root a is the White node. The value of the nodes is the number next to each node. Node i is White, so its value is max (3,-2) = 3, node d is Black, so its value is min(2, 3, 4) = 2.

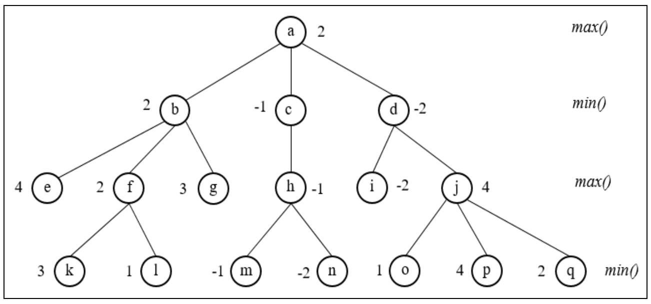


Fig 3: Assigning values of game tree definitions

Assinng the values to the nodes is done by the recurrent functions MaxVal and MinVal. The MaxVal function determines the value for the White nodes, the MinVal function determines the value for the Black nodes.

```
function MaxVal(u){
if u is the ending node then
MaxVal(u) f(u)
else MaxVal(u) ← max{MinVal(v) | v is
a sub-node of u;
return ...;
}
//----------------------------------
-----------------
function MinVal(u){
if u is the ending node then
MinVal(u) f(u)
else MinVal(u) ← min{MaxVal(v) | v is
a sub-node of u;
return ...;
}
```

In the above recurrent functions, f(u) is the value of the play-off function at the ending node u. The following procedure shows a move selection for White at node u. In Minimax(u,v), v is the variable storing the state that White has chosen the movement from u.

```
function Minimax(u, v){
val ← -∞;
for each w is a sub-node of u
if val <= MinVal(w){
val ← MinVal(w);
v ← w};
return ...;
}
```

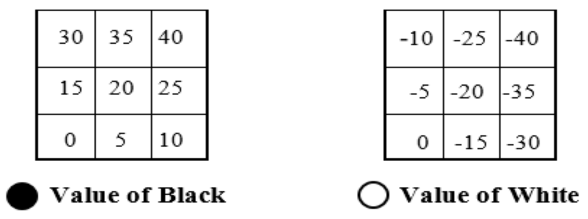
The move selection step above is called the Minimax strategy because White has already selected a move that leads to a sub-node whose value is the max in the values of the sub-nodes, and Black responds with a move to a node whose value is the min in the values of sub-nodes.
Minimax algorithm is a depth search algorithm, in this situation, we have installed Minimax algorithm by recurrent functions. Readers, please write a non-recurrent procedure to implements this algorithm.
Theoretically, the Minimax strategy allows us to find the optimal move for White. However, it is not practical, we do not have enough time to calculate the optimal moves due to the Minimax algorithm requires us to consider all the nodes of the game tree. In good games, the game tree is extremely large. For example, for chess, taking into account only a depth of 40, the game tree already has about 10^{120} nodes. If the tree has height m, and at each node is b move, then the time complexity of Minimax algorithm is O (b^m).
In order to quickly find a good (non-optimal) move instead of using the play-off function and consider all the possibilities leading to the ending states, we use the evaluation function and only consider a part of the game tree.

3.1 Evaluation Function

The eval evaluation function corresponds to each u state of the game with a numerical value eval(u), which is an estimate of the "advantage" of u state. The more advantageous the u state is for White, the larger the positive number of the eval(u) is; The more advantageous u state is for Black, the smaller the negative of the eval(u) is; eval(u) ≈ 0 is a disadvantageous state for everyone.
The quality of the chess program depends a lot on the evaluation function. If the evaluation function gives us an incorrect estimate of the states, it can guide us to a good state, however, actually we are at a disadvantage. Designing a good evaluation function is a difficult task, requiring us to consider many factors: the remaining chess pieces of the two sides, the arrangement of those chess pieces, etc. There is a contradiction between the accuracy of the evaluation function and its computational time in this situation. The exact evaluation function requires a lot of computational time, and the players are limited by the time when they make a move.
The following simple evaluation function for chess is proposed: Each type of chess piece is assigned a numerical value that corresponds to its "strength". For instant, each White (Black) pawn is given 1 (-1), a White (Black) bishop or knight is given 3 (-3), a White (Black) castle is given 5 (-5) and a White (Black) queen is given 9 (-9). Taking the sum of the values of all the pieces in a state, we get the value of that state. Such an evaluation function is called a weighted linear function because it can be expressed as:

$$w_1 S_1 + w_2 S_2 + \cdots w_n S_n \quad (1)$$

In which, wi is the value of each type of piece, and Si is the number of pieces of that type. In this assessment, we did not think to the arrangement of the pieces and the relationships between them.
Now a way of evaluating states in the Dodgem game is proposed. Each White (or Black) piece in a position on the chessboard is given a corresponding value, as shown in Fig 4.



30	35	40
15	20	25
0	5	10

● Value of Black

-10	-25	-40
-5	-20	-35
0	-15	-30

○ Value of White

Fig 4: Evaluation of the position of the pieces in the game Dodgem

In addition, if White directly blocks Black piece, it gains 40 points, if it is an indirect obstruction, it gains 30 points (See Fig 5). Similarly, if Black directly blocks White, it gains -40 points, if it is an indirect obstruction, it gains -30 points.

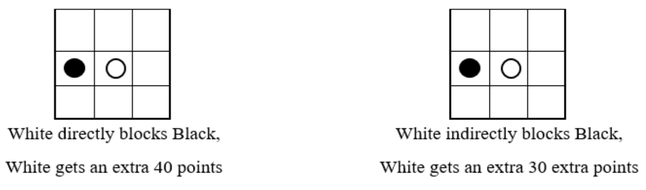

White directly blocks Black,
White gets an extra 40 points
White indirectly blocks Black,
White gets an extra 30 extra points

Fig 5: Evaluation of the correlation between White and Black pieces in the game Dodgem

Applying the above rules, we can calculate the value of the state on the left of Fig 6 is 75, the value of the state on the right of the figure is -5.

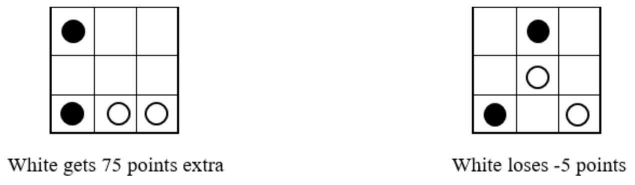


Fig 6: Evaluation of the correlation between White and Black pieces in the game Dodgem

In the above assessment, we have considered the position of the pieces and the relationship between the pieces.
In order to limit the search space in a simple way is to determine the movement for White at u, we only consider the game tree at root u to some height h. Applying the Minimax procedure to the game tree at root u, the height h and using the value of the evaluation function for the leaves of that tree, we will find a advantageous move for White at u.

3.2 Alpha-Beta Cut Method

Although we limit the search space to the game tree at root u with height h, the number of nodes of this game tree is also very large with h ≥ 3 in the Minimax searching strategy, a way that help to find a beneficial move for White at state u. For example, in chess, the average of branch factor in the game tree is about 35, required time to make a move is 150 seconds when your program on a conventional computer can only consider nodes in the depth of 3 or 4. A chess player at the medium level can precalculate 5, 6 moves or more, and thus your program will meet the need of the beginner level.
When evaluating node u to depth h, a Minimax algorithm requires us to evaluate all nodes of the tree at root u to depth h. However, we can reduce the number of notes that need to be evaluated without affecting the evaluation of node u. The alpha-beta cut method allows us to remove branches that are not necessary for the evaluation of node u.
The ideal of alpha-beta cut method is as follows: Remember that the Minimax searching strategy is a depth searching strategy. We can assumes that in the search process we go down node White a, and node a has a brother v has been evaluated. And we also assumes that father of node a is b and b has a brother u already evaluated, and father of b is c (Fig 7). And we have the min value of node c (White node) is the value of u, the max value of node b (Black node) is the value of v. Therefore, if eval(u) > eval(v), we do not need to move down to evaluate node a without affecting node c. In other words, we can cut off the subtree at root a. We have the similar argument for the case that a is a Black node, in this case if eval(u) < eval(v) we can also cut the subtree at root a.

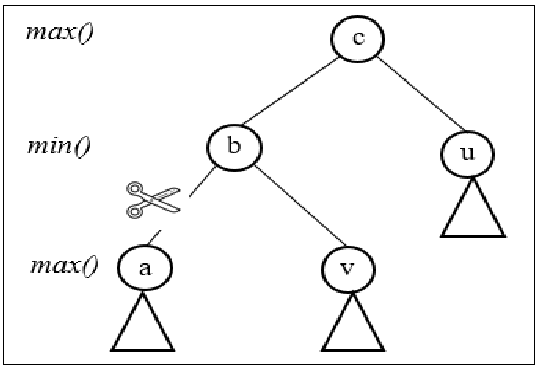


Fig 7: Cut off the subtree at root a, if eval(u)>eval(v)

In order to install the alpha-beta cut method, for nodes lying on the path from the root to the current node, we use the parameter α to record the maximum value of the evaluated sub-nodes of a White node, and the parameter β records the minimum value of the evaluated sub-nodes of a Black node. The values of α and β will be updated during the searching process. α and β are used as local variables in the functions MaxVal(u, α, β) (function to determine the value of the node Black u) and Minval(u, α, β) (the function to determine the value of the node Black u).

```
function MaxVal(u, α, β){
if u is the leaf of the limitted
tree or u is the ending node
MaxVal ← eval(u)
else for each node v is a sub-node
of u
{α ← max[α, MinVal(v, α, β)];
// Cutting out the subtrees from the
remaining node v
if α ≥ β exit};
MaxVal ← α;
}
//----------------------------------
------------------
function MinVal(u, α, β){
if u is the leaf of the limitted
tree or u is the ending node
 MinVal ← eval(u);
else for each node v is a sub-node
of u
{β ← min[β, MaxVal(v, α, β)];
// Cutting out the subtrees from the
remaining node v
if α ≥ β exit};
MinVal ← β;
}
```

The algorithm searching for the move of White uses the alpha-beta cut method, installed by the procedure Alpha_beta(u,v), where v is the parameter recording the node that White needs to reach from u.

```
procedure Alpha_beta(u,v){
α ← -∞;
β ← ∞;
for each node, w is a sub-node of u
if α ≤ MinVal(w, α, β)
{α ← MinVal(w, α, β); v w;}
}
```

The game tree at root u (White node) limitted by height h = 3, as denoted in Fig 8. The number next to the leaves is the value of the evaluation function. Applying the Minimax strategy and the cut method, we determine the best move for White at u, which leads to the node v with the value 10. Next to each node, we also give the value of the parameter pair (α, β). When calling the MaxVal and MinVal functions to determine the value of that node. The removed branches are shown in Fig 8.

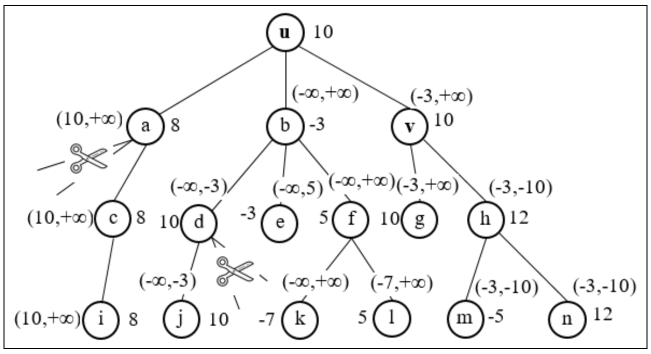


Fig 8: Determination of node values using Alpha – Beta cut method

4. Conclusion

In this paper, the authors have introduced the algorithm for solutions in fighting game with a search space represented by a search tree. The game used to illustrate the algorithm is the Doegem game. In order to reduce the search space, make the representation easier and speed up searching for solution, the Alpha - Beta cut method has been proposed. From illustrative examples with specific values, the effectiveness of the method mentioned is clearly demonstrated. Basing on this result, we are going to install specifically an application to make the test more thoroughly.

5. Acknowledgment

This paper was implemented by authors who are the lecturers of Thai Nguyen University of Technology (TNUT). Thank TNUT for your help and support to complete this study.

6. References

1. Charniak E. Introduction to artificial intelligence, Pearson Education India, 1985.
2. Nilsson NJ, Nilsson NJ. Artificial intelligence: A new synthesis, Morgan Kaufmann, 1998.
3. Bratko I. Prolog programming for artificial intelligence, Pearson education, 2001.
4. Korb KB, Nicholson AE. Bayesian artificial intelligence, CRC press, 2010.
5. Mitchell RS, Michalski JG, Carbonell TM. An artificial intelligence approach, Springer, Berlin, 2013.
6. Genesereth MR, Nilsson NJ. Logical foundations of artificial intelligence, Morgan Kaufmann, 2012.
7. Nilsson NJ. Principles of artificial intelligence, Morgan Kaufmann, 2014.
8. Russell S, Norvig P. Artificial intelligence: A modern approach, 2nd Edition, Prentice-Hall, 2003.
9. Ginsberg M, Morgan Kaufman, Essentials of artificial intelligence, 1993.
10. Ginsberg M. Essentials of artificial intelligence, Newnes, 2012.
11. Ahlquist JB, Novak J. Game Artificial Intelligence, Thomson, 2008.
12. Barr A, Feigenbaum EA. (Eds.). The handbook of artificial intelligence. Butterworth-Heinemann. 2014; 2.
13. Wolfgang Ertel, Introduction to Artificial Intelligence (Second Edition), Translated by Nathanael Black with illustrations by Florian Mast, Springer International Publishing AG, 2017.