CONNECT4

A PREPRINT

João Pedro Gonçalves de Aguiar
Faculty of Science
University of Porto
up201606361@fc.up.pt

Pedro Sobral da Costa
Faculty of Science
University of Porto
up201606361@fc.up.pt

April 6, 2019

ABSTRACT

In this paper, we explore the project of Connect4 we have developed with 2 game options: playing against the computer and playing among humans. We explore the first option here. Our goal was to find the best algorithm, in terms of speed and best plays, to be used by the computer. To explore this problem, we have have used 3 different choice strategies: MiniMax, Alpha-Beta (Pruning) and Monte Carlo Tree Search. MiniMax and Alpha-Beta work similarly, although the second is faster. They make the faster and better choices for the computer. Monte Carlo Tree Search is playing kind of "randomly" so it may not give us the better plays every time.

1 Introduction

Connect4 is an example of an *Adversial Game* between two players. This kind of games can be interpreted as a strategy of different moves, with the goal of reaching a position where it is possible to win. These types of interactions are studied by game theory.

From the point of view of game theory and artificial intelligence, the interest is quite large: "How fast can we win?". Since in each state of the game, there are a large number of possible moves, it is a challenge to choose, quickly as possible, which one is the best to get a faster victory.

2 Adversial Search Algorithms

Although we have changed, not too much, the algorithms described below, here we describe their raw behavior. See [1] for a more detailed description of them.

2.1 MiniMax

This algorithm finds the best play expanding the initial game state till a given depth. At each level, it chooses the best play for the current player of the level. This choice is made with an evaluation of the current state of the game. This evaluation function is explained in Section 4.

The use of the minimum or maximum type in each level of the tree is due to the existence of two players: the caller has positive punctuation and the other has a negative one. So, at the levels of the player who is calling the MiniMax, it chooses the state with maximum value. In the opponent's positions it chooses the minimum value.

It uses a simple recursive computation of the minimax values of each successor state. The recursion runs all the way down to the leaves of the tree, and then the minimax values are propagated back through the tree as the recursion returns.

2.2 Alpha-Beta (Pruning)

This one is similar to the previous one, but in this one there are some branches of the trees that are cut off (Pruning), because the algorithm "decided" they are unnecessary, increasing the speed of response. To implement this pruning there will exist two more variables, alpha and beta, which respectively, keep the maximum and minimum level.

At the maximum level, the greatest value is chosen among the successors of the node and the value of alpha is updated. If the evaluation (v) is such that

$$v > \beta,$$

the cut is also performed. In the case of a minimum level, the lowest value is chosen between the successors of node and the value of *beta* is updated. If

$$v < \alpha, \tag{1}$$

the cut is made.

2.3 Monte Carlo Tree Search

Although MiniMax and Alpha-Beta can solve some problems, they may not be so useful if we have a lot of possible states along the game, it would get tedious to wait for the answer. That's why we introduce MCTS. It is divided in 4 steps:

- Selection: it expands the root node till it finds the last node before the leaf one.
- Expansion: takes this last node and expands it.
- Simulation: it simulates a random play over this expanded node and stores the UCB evaluation from that board.
- Backpropagation: propagates that value over the visited nodes (in selection). So, this value is, for now, the best.

  After this, the process is repeated the number of times we want (the higher, the better) in order to find an even better move.

3 Connect 4

Connect 4 is a two-player game in which each player chooses a different colour and then they take turns dropping some coloured disc in a 6x7 grid. The goal is to try to obtain 4 consecutive positions with the same colour, in the following orientations: vertical, horizontal and diagonal.

There is a limited number of moves in each instance of the game. Some that provide the evolution to a position more winning and others less. In order to make this evaluation, we use the algorithms MiniMax, Alpha-Beta and Monte Carlo Tree Search with an evaluation function. Inside these algorithms we work with an evaluation function that makes the following evaluation: adds 16 points if the move is made by the Computer and adds -16 in the case of the Human. Then evaluate each set of four elements, where all possibilities are considered.

A win by the Computer has a value of +512, while a win by the Human has a value of -512. The rules for other segments are as follows:

- -50 for three Os, no Xs,
- -10 for two Os, no Xs,
- -1 for one O, no Xs,
- 0 for no tokens or mixed tokens,
- 1 for one X, no Os,
- 10 for three Xs, no Os,
- 50 for three Xs, no Os,

Besides this, we've added some other conditions to make better choices, explained below.

4 Adversial Searches applied to Connect4

4.1 Programming language

We chose to work with *Java* because it is object-oriented designed so it becomes easier to understand the organization of the code and it is more meaningful.

4.2 Data structures

ArrayList We have used this static structure from the Java API to store the generated nodes (ChildBoards). They were a good choice because we had a function, inside the children generation function, that tells us if it is possible to generate a certain child. So this ArrayList has always a fixed size.

4.3 Algorithm organization and constraints

We've divided the classes as:

- Connect4 - requests input, creates the *game object* - gameboard and players - and chooses the algorithm to be used by the computer, if that's the case.
- GameBoard - Works with the *gameboard object*.
- Player - Creates *object* instances of the two players (withsymbols and names, if that's the case).
- TypesOfSearch - Different search algorithms.
- Global - Has some variables, used by those 3 algorithms, that need to remain constant along different calls of the algorithms' functions.

For the MCTS algorithm we've used 10000 iterations over it.

We have also implemented some useful functions inside the *GameBoard class*:

- endGame() - Checks if the current state is final, that means, if someone has already win or if it is a draw.
- evaluatefunc() - Calculates the value of the current board (only used playing against the computer).
- evaluateUCB() - Calculates a different value of the current board for the MCTS algorithm (only used playing against the computer).

We have also changed the values returned from the evaluation function in each step of the algorithm. It returns the evaluation value minus the depth of that step so we can always get the faster move to the best evaluated board. Besides this, so the computer may always win and stop us from winning, we have also inserted some break conditions and the *StopOP function* which tells us, in the case the human could win the game after the computer's move, if it's possible to stop him or if we can play in some other column.

5 Results

We've made some performance curves for the expanded nodes of MiniMax algorithm and Alpha-Beta, as follows in Section 7 and also time performance curves of MiniMax, Alpha-Beta and Monte Carlo. As we can see, Alpha-Beta is the fastest one without loss of performance and the cheapest one, in terms of memory, compared to MiniMax.

6 Overview

As we can see, MiniMax can be very slow, and it plays the same way as Alpha-Beta does, so we might choose the seconde if we want a fast and accurate algorithm. Monte Carlo is also a good approach for this kind of problems, but gains importance when it comes to several branching levels. In this case, Alpha-Beta is a better choice.

7 References and results data

References

[1] Stuart Russel and Peter Norvig. In *Artificial Intelligence 3rd Edition, 2016, pages 75–108. IEEE, 2014.*

Figure 1: Performance curve of MiniMax

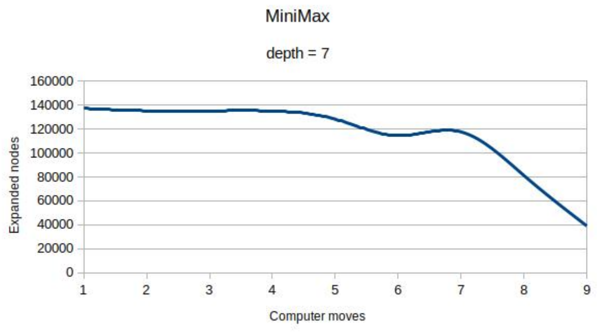


Figure 2: Performance curve of Alpha-Beta

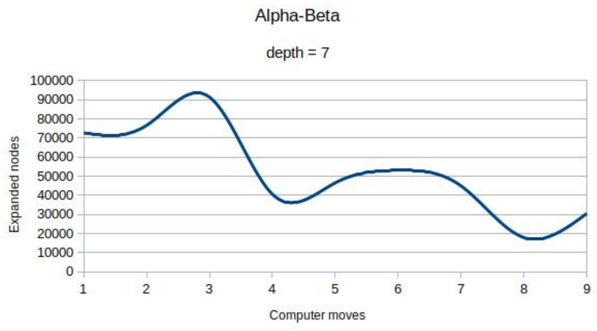


Figure 3: Time performance curve for MiniMax

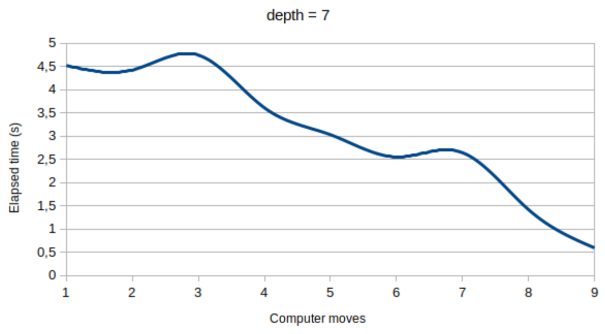
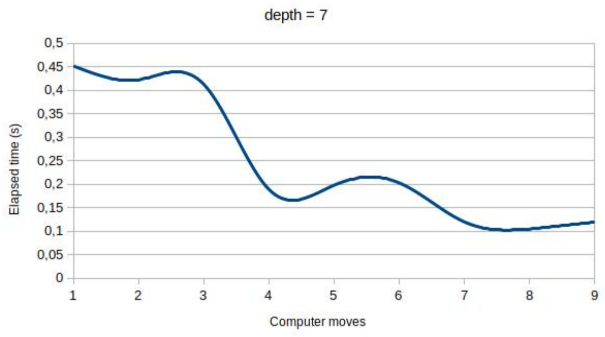


Figure 4: Time performance curve for Alpha-Beta


depth = 7
Elapsed time (s)
Computer moves