Dynamic connect-3 report

Romain Bazin

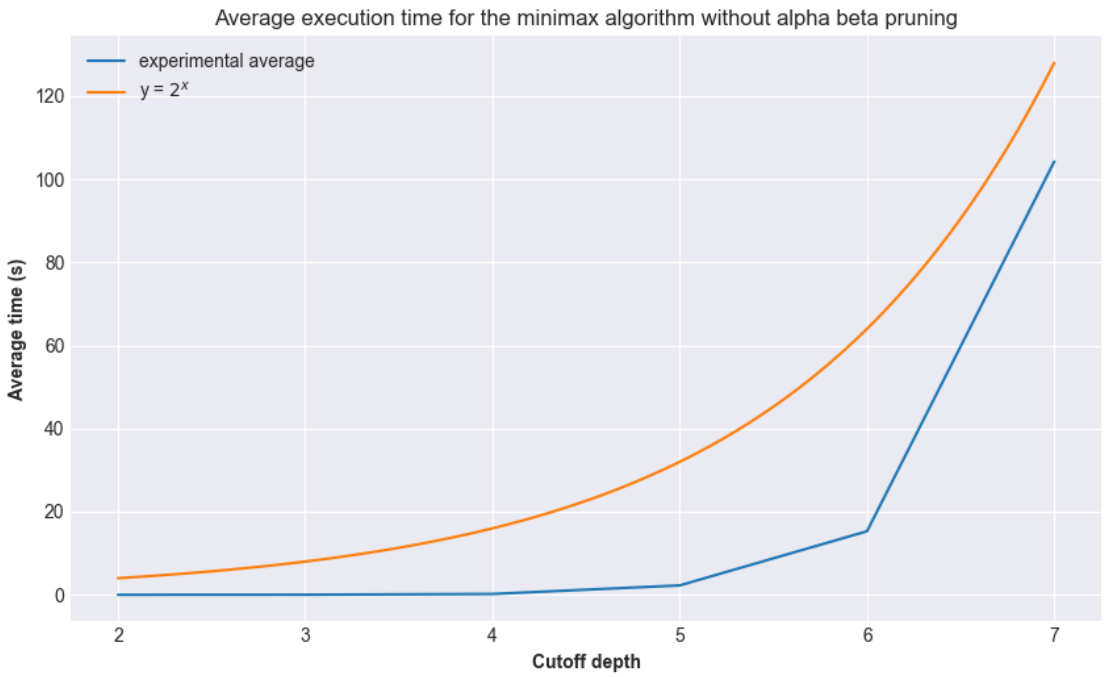
Part I

The goal of this first part is to build a functional agent, able to play against a human and another agent via a TCP connection on the game of dynamic connect-3.

We compare the performance of the agent with and without pruning during the minimax algorithm by computing the average of the computation time necessary to explore the tree of possible states until a certain cutoff depth.

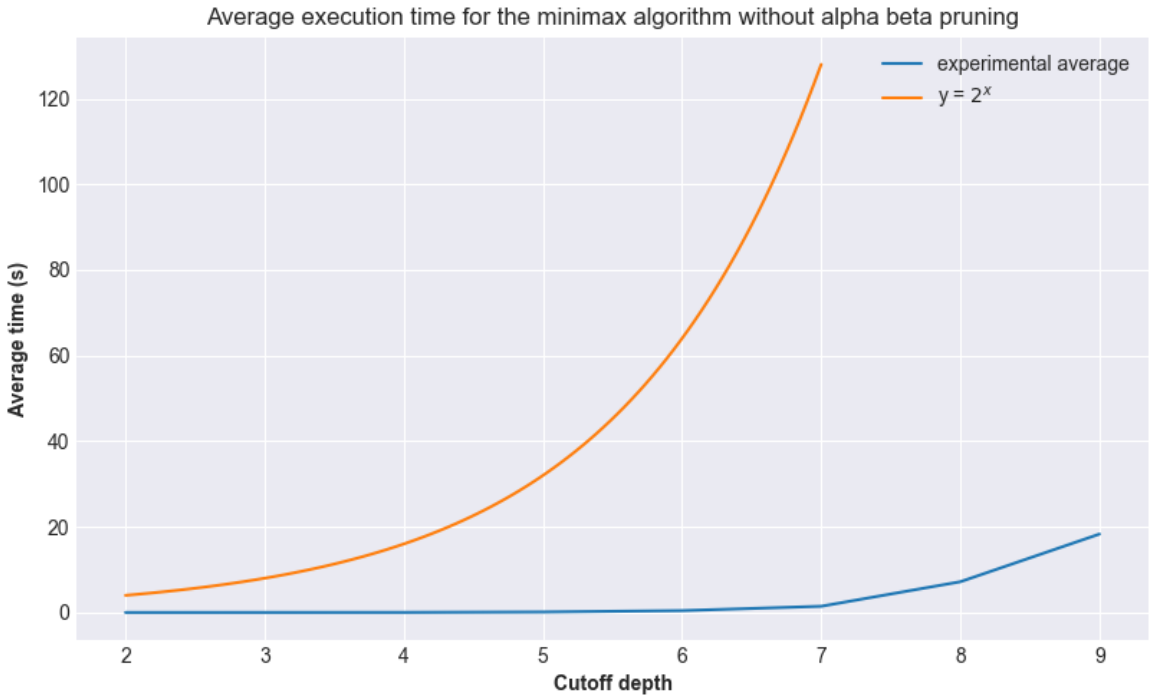
Without alpha-beta pruning :

Depth cutoff	2	3	4	5	6	7
Average computation time (s)	0.0044	0.03131	0.2051	2.2640	15.2903	104.24

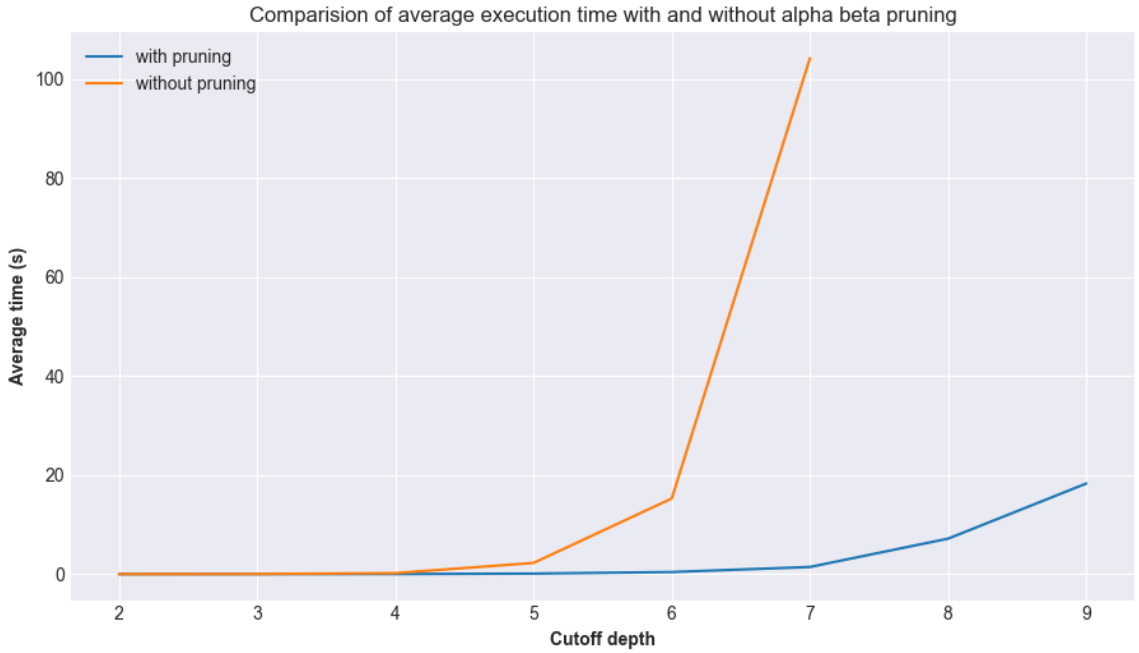


With alpha-beta pruning :

Depth cutoff	2	3	4	5	6	7	8	9
Average computation time (s)	0.0025	0.0108	0.0229	0.1158	0.4242	1.447	7.1743	18.311



One against the other :



We see that pruning drastically increases the computation speed of the minimax search, allowing us to choose a cutoff depth of 8 instead of 5, while still being under the 10s limit.

By going deeper into the tree, the agent can, in theory, predict more turns ahead of the opponent, thus choosing the best actions possible for its own outcome.

Note : all these tests have been conducted with the basic heuristic
$h(s) = 2runsagent - 2runsopponent$

Part II

We mentioned in part I how important it is to go deeper than one's opponent into the tree of possible states in order to predict the outcomes ahead of him, thus choosing the best actions possible to reach these outcomes.

However, going deep into the tree is far from being the only important parameter. Indeed, the accuracy of the cutoff evaluation, computed by the heuristic function, is what evaluates if a state is good or not. Thus, it determines the whole strategy and goal of the agent. As a matter of fact, the better the heuristic, the more accurate the evaluation will be, leading to better outcomes.

However, computing a *good* heuristic is not that easy, especially when we take into account the time limit to play each turn. Indeed, computing a better heuristic will increase the overall computing time, hence reducing the depth possible reachable by the minimax algorithm.

Some possible improvements for the heuristic are listed there but we will have to ponder if using it is worth the computation cost, by doing some experiments.

All of the following additions to the basic heuristic will be computed positively for our agent and negatively for the opponent.

- Consider if there's a possible win in 1 or 2 turns
- Calculate the number of pawns blocked
- Calculate the number of possible moves
- Calculate the distance from the center of all pawns (I assume the closer to the center, the better)
- Calculate the total distance between pawns of same color
- If we are able to identify good or bad patterns, we may add them in the heuristic (it was not implemented, but this is where feedback from a good player would have been interesting)

Part III

As the whole program has been developed so it's generalizable to other grid sizes, we just need to change the initiate_grid function so that it can create a grid of size 7x6.

```
def initiate_board(w, h):
    board = -1 * np.ones((h, w))
    if (h, w) == (6, 7):
        board[0 + 1, 0 + 1] = 0
        board[0 + 1, w - 1 - 1] = 1
        board[1 + 1, 0 + 1] = 1
        board[1 + 1, w - 1 - 1] = 0
        board[2 + 1, 0 + 1] = 0
        board[2 + 1, w - 1 - 1] = 1
        board[3 + 1, 0 + 1] = 1
        board[3 + 1, w - 1 - 1] = 0
    else:
        board[0, 0] = 0
        board[0, w - 1] = 1
        board[1, 0] = 1
        board[1, w - 1] = 0
        board[2, 0] = 0
        board[2, w - 1] = 1
        board[3, 0] = 1
        board[3, w - 1] = 0

    return board
```