ECSE 526-Artificial Intelligence

Assignment 1-Dynamic Connect 4

Alexandre Coulombe - 260801407

January 28, 2020

1 Total number of states explored

The three game states used for the evaluation of the number of states explored using different depth configurations are described in figure 1.

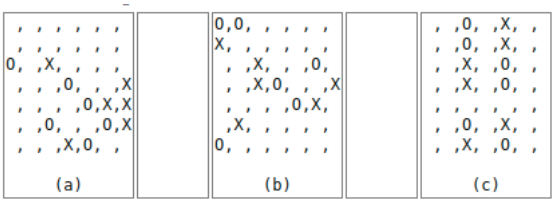
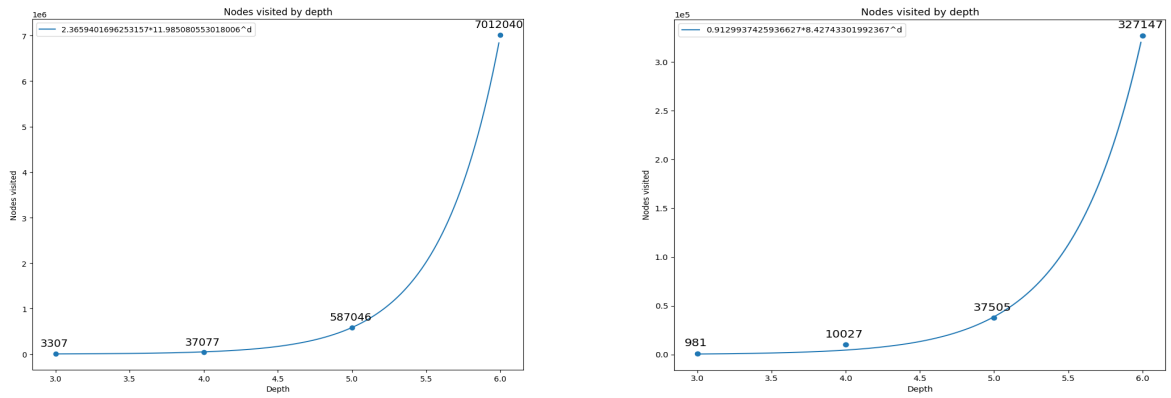


Figure 1: Game states used for the evaluation of the number of states explored using different depth configurations

The data collected about the number of states explored by the program using various depths and the algorithms minimax and alpha beta pruning can be found in figures 2(a), 2(b), 3(a), 3(b), 4(a) and 4(b). As expected, the number of states visited by the minimax algorithm is much greater than that of the alpha beta pruning algorithm. This is due to alpha beta pruning being an improvement of minimax by using the idea of pruning to not explore states that do not impact the decision on the final move selected. This is because, when the player has found a better alternate move prior, a state will never be reached. So when enough information about a state is found to determine it will not be reached, it can be pruned. Thus, the number of nodes visited changes depending on the depth of the search and the current game state.



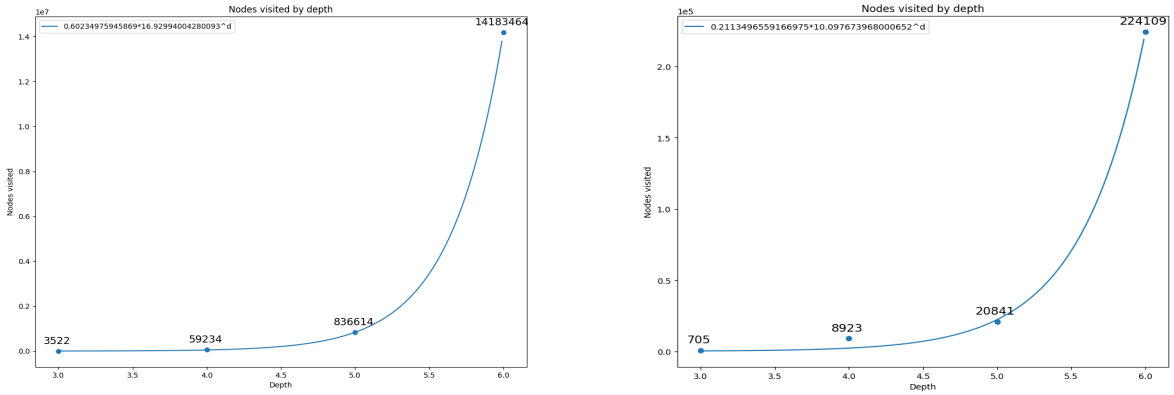
(a) Minimax

(b) Alpha Beta Pruning

Figure 2: Nodes visited by depth of game state A using Minimax and Alpha Beta Pruning

2 Depth cutoff to the number of states visited formula

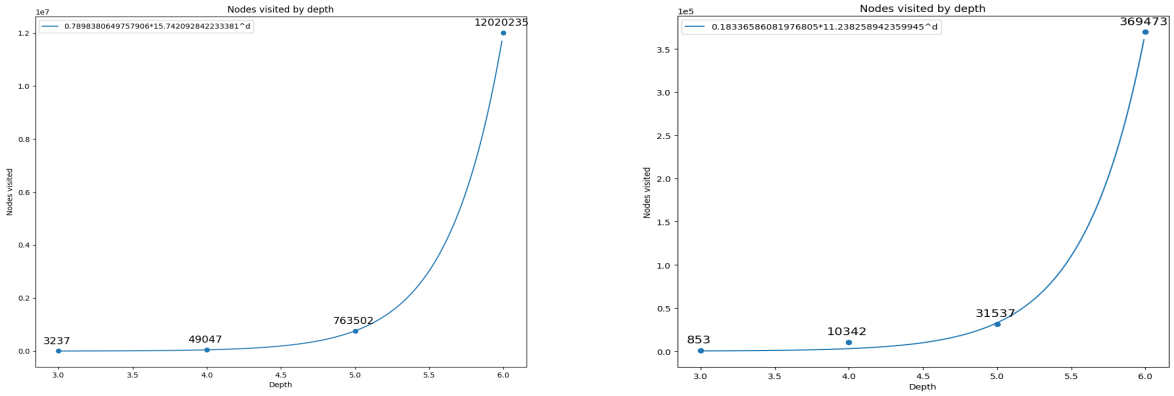
Due to the exponential nature of the data points, the formula $states = a * b^d$ is used to estimate the relationship between the depth of the search and the number of nodes/states visited by the program. A curve fit was



(a) Minimax

(b) Alpha Beta Pruning

Figure 3: Nodes visited by depth of game state B using Minimax and Alpha Beta Pruning



(a) Minimax

(b) Alpha Beta Pruning

Figure 4: Nodes visited by depth of game state C using Minimax and Alpha Beta Pruning

performed on each data set to find the values of the function that has the least squares error to the values of the depth and corresponding visited nodes. For the minimax algorithm, the value for a should be 1 in theory because there is only one starting state. The values found for the branching factor, b, are 11.985 for game state A, 16.923 for game state B and 15.742 for game state C. This gives an average of 14.883, which means, on average, each state has 14.883 possible moves. This would give the formula $states = 14.883^d$ for the number of states visited for a certain depth, d, by the minimax algorithm.

The alpha beta pruning data also shows an exponential form to it, so the estimated formula will also use the estimate $states = a * b^d$. A curve fit was performed on each data set to find the values of the function that has the least squares error to the values of the depth and corresponding visited nodes, as it was for minimax. For the alpha beta pruning algorithm, the value for a should be 1 in theory because there is only one starting state. The values found for the branching factor, b, are 8.4274 for game state A, 10.0977 for game state B and 11.238 for game state C. This gives an average of 9.921, which means, on average, each state has $14.883 - 9.921 = 4.962$ possible moves and pruned. This would give the formula $states = 9.921^d$ for the number of states visited for a certain depth, d, by the alpha beta pruning algorithm. In summary, the number of state can be estimated as in (1), where d represents the depth.

$$states = \begin{cases} 14.883^d, & \text{if minimax} \\ 9.921^d, & \text{if alpha beta pruning} \end{cases} \tag{1}$$

3 Generation order of new states

The number of states explored highly depends on the order in which you generate new states during the search. By having the best successors be generated first, the other worse successors could be pruned. By pruning the

worse successors, we do not need to waste resources, such as time and memory, examining those states, which would not have an impact on the final decision for the selected move. In theory, by generating the best successor states first, the alpha beta pruning algorithm would visit $O(b^{m/2})$ nodes, while a more random ordering of the successor states would look be around $O(b^{3m/4})$ nodes visited. So by ordering the moves so that the best successors are generated first would allow the program to prune the worse successors, save resources, and allow the program to do deeper searches for the same amount of time.

The effect of the move ordering can be observed by using three ordering forms on the same game state: unsorted moves, sorted moves, and random moves. The game state used to produce results is game state A in figure 1 and is using a depth search of 4. Using unsorted moves, the number of nodes visited is 10027 nodes. Using sorted moves, the number of nodes visited is 5798, which is nearly half as many. Using a random order, the number of nodes visited were 11465, 8875, and 10279 nodes, which are all comparable to the unsorted order of the moves. So, by using move ordering, we can see that less nodes are visited during the search.

4 Heuristic evaluation function

The heuristic evaluation function used by the program is a combination of two heuristics. The first heuristic is utility for the amount of pieces aligned, since the goal to win the game is to align four of your own pieces before your opponent does. The heuristic gives value based on the amount of pieces aligned. The value of two pieces being aligned is 1 point, the value of three pieces being aligned is 3 points and the value of four pieces being aligned is 1000, so that the agent will do the sequence to get the win. This heuristic gives utility to the program when it aligns pieces and a higher utility when it aligns even more pieces. This heuristic is to push the program to reach the goal of the game so that it is able to win against its opponent. If it were not included, the program would not have any incentive to align pieces, which is the entire point of the game of dynamic connect 4.

The other heuristic used is the environment central dominance heuristic. The heuristic gives higher and higher value to pieces that get closer and closer to the center of the environment, in this case the center of the dynamic connect 4 game board. This gives the program incentive to move its pieces to the center of the board, and closer to each other in consequence. Due to the pieces being separated at the start of the game on opposite sides of the game board, three on the left and three on the right, in order to align four pieces, at least one piece must traverse the board to reach the other pieces to align four pieces. However, by controlling the center of the board, more of your own pieces will be closer to each other, which allows the program to find states which align the pieces. The values of the tiles are saved in a 2D array assigning higher and higher value to tiles getting closer and closer to the center, with the center having the highest value.

By combining both of the heuristics, the program will choose actions that have the highest utility, which means moves that place pieces towards the center of the board and align pieces that are near each other will be played. However, if we consider the utility of the opponent's moves, we can find states in which we have a higher utility and the opponent has lower utility and perform the sequence of actions to reach those states. This makes the final heuristic $playerheuristic - opponentheuristic$, which allows the player to play moves that benefit it and disadvantage the opponent.

5 Simple vs Complex evaluation function

Due to the exponential nature of the search, it would be favorable to have a more complex evaluation function and have a shallower depth search. By having a simple evaluation function, it is true that the program can reach deeper depths of the search, but each depth would take exponentially longer to complete the search. The simple evaluation function would be fast, but wouldn't get enough information about the state of the game, leading to more ignorant decisions. By having a more complex evaluation function, more information can be gathered about the states which would allow the program to make better decisions. The increase in time of the computation of the evaluation function compared to the time it requires to perform the simple evaluation function is not comparable to the amount of additional time required to search an extra depth. In addition, the amount of memory required to reach deeper depths, however the memory usage grows linearly and not exponentially like time does. This may not be an issue for powerful computers with a lot of memory, but it is a consideration when developing for a low-power computational device. Thus, a more complex evaluation function with a shallower depth search would be more beneficial than a deeper search with a simpler evaluation function.