CS F407
ARTIFICIAL INTELLIGENCE

ASSIGNMENT 2

Connect 4 AI with Minimax Algorithm

Kenz Abdulla

2021A7PS2664G

Evaluation Functions

Normal Distribution

A rudimentary evaluation function was implemented first, considering that a cell in the centre of the connect4 table leads to better chances of winning than cells in the periphery. Hence, the evaluation function will look like a normal distribution centred around the mean position (Fig 1). The value of a game state will be the sum of values at coordinates where the cells are empty and subtracting the values at coordinates where cells are filled with opposite player's tokens.

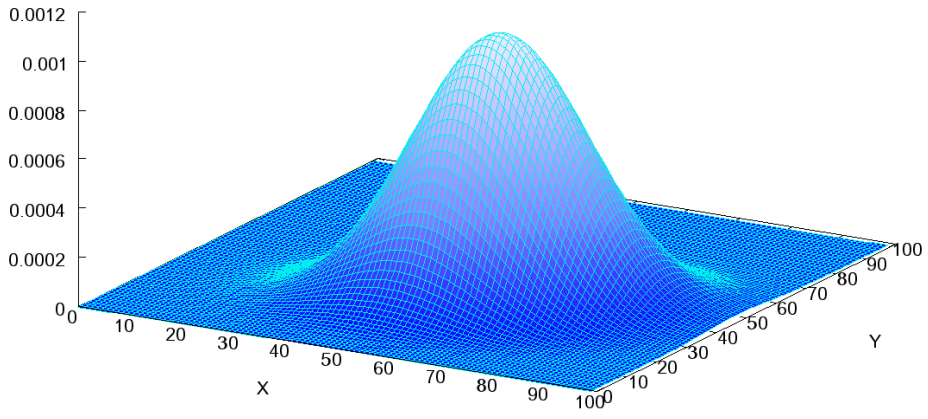

Fig. 1 (Not to scale)

This is not a complete evaluation function and doesn't translate to a high percentage of wins for depth three but gives more than half of the wins for depth five (Fig. 2a and 3a).

Consecutive Count

A player wins the game if he connects four tokens of his colour in a row, column or any positive or negatively sloped diagonal. As the next evaluation function, I have considered taking the counts of consecutive occurrences of the player's token. Consecutive four, three and two occurrences are assigned decreasing scores, respectively. This evaluation function improves the accuracy to 42 wins out of 50 for a depth of three and 45.5 for a depth of five.

Windows of 4

In the Consecutive Count function, all occurrences are considered, even ones with opposite player's tokens on either side of three consecutive GameTreePlayer's tokens. Effectively blocking us from winning. Hence, in this new function, all windows of length four are

considered, and a score is assigned to a board state based on the number of tokens of the player in each window. Each of the other cells in the window must be zero. Since four tokens is a win state, it adds a much higher value to the score. Windows with 3 and 2 tokens add smaller values, which have been tuned to obtain maximum performance (Fig. 2a and 3a).

Windows of 4 + Preference to the columns near the centre

Using the above evaluation criteria and modifying it to give a higher score to moves that place a token in the column in the centre leads to marginally higher performance (Fig 2a and 3a).

GameTreePlayer - MyopicPlayer

Using the previous evaluation function, which considers windows and prefers the centre column with respect to both GameTreePlayer and MyopicPlayer and subsequently taking their difference as the final evaluation function leads to better performance in the range of 99.5% (for depth of 5) and 97.5%(for depth of 3) (Fig. 2a and 3a). This is a near-perfect agent.

For the MiniMax search with depth 5, modifications to the 4-window evaluation function only lead to a marginal increase in performance. The difference in performance is better observed with a depth of 3. The average number of moves is higher for the normal distribution evaluation function and comparable for the others (Fig. 2b and 3b). The modifications don't lead to improvement in the number of moves.

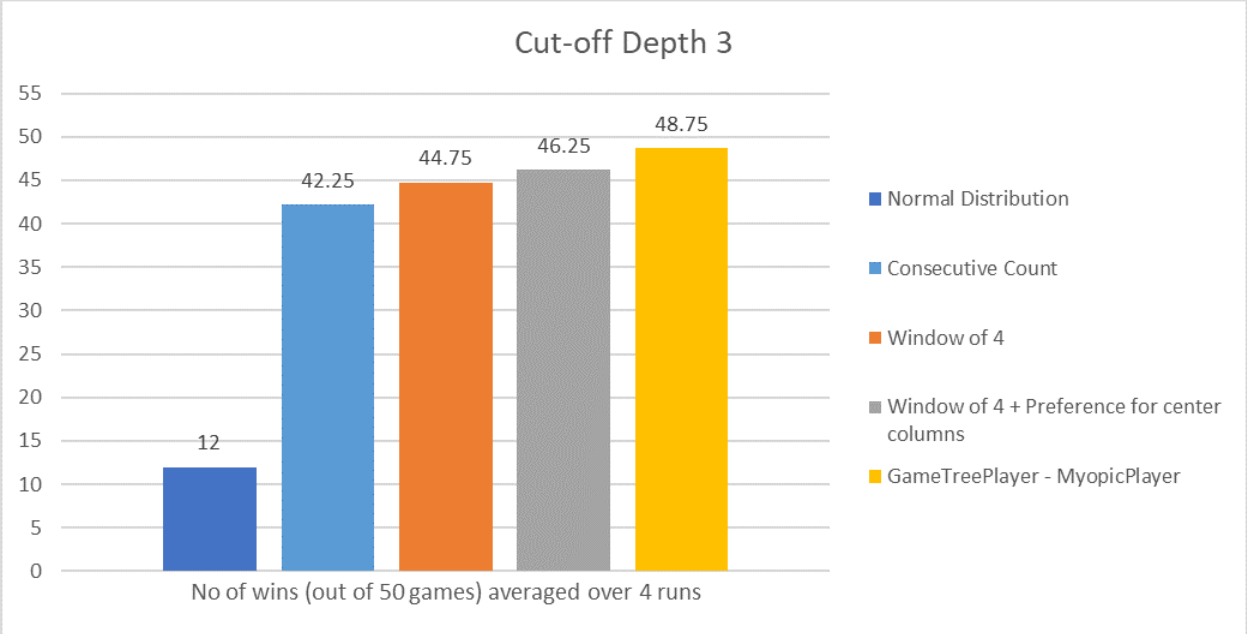

Fig. 2a

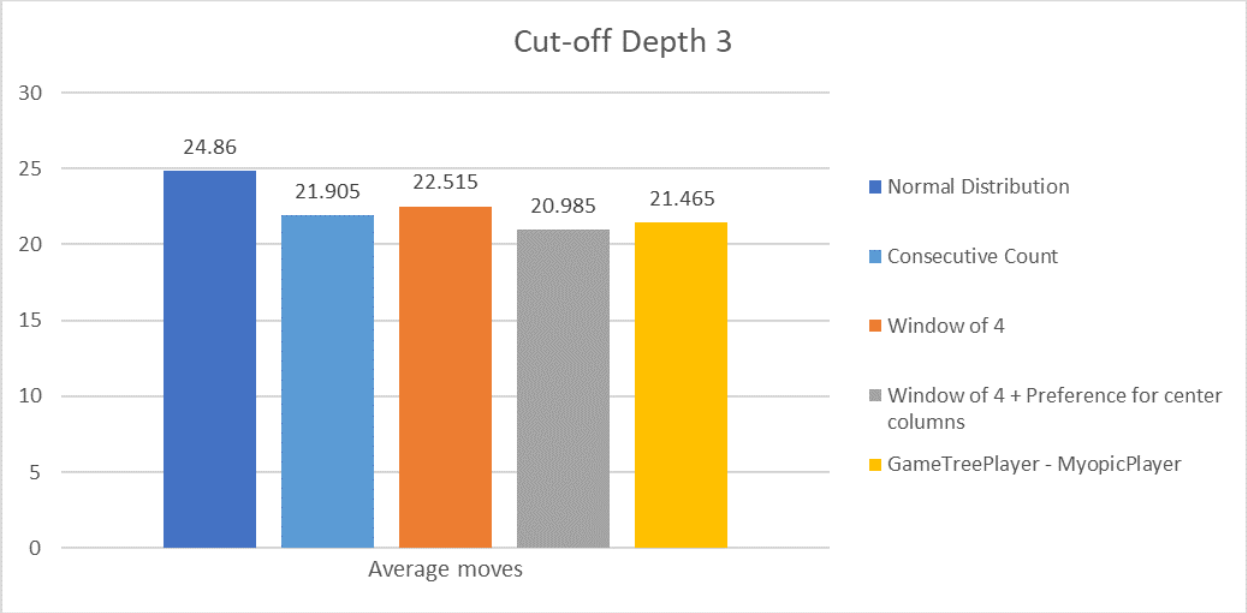
Cut-off Depth 3
30
25
20
15
10
5
0
24.86
21.905
22.515
20.985
21.465
Average moves
Normal Distribution
Consecutive Count
Window of 4
Window of 4 + Preference for center columns
GameTreePlayer - MyopicPlayer

Fig. 2b

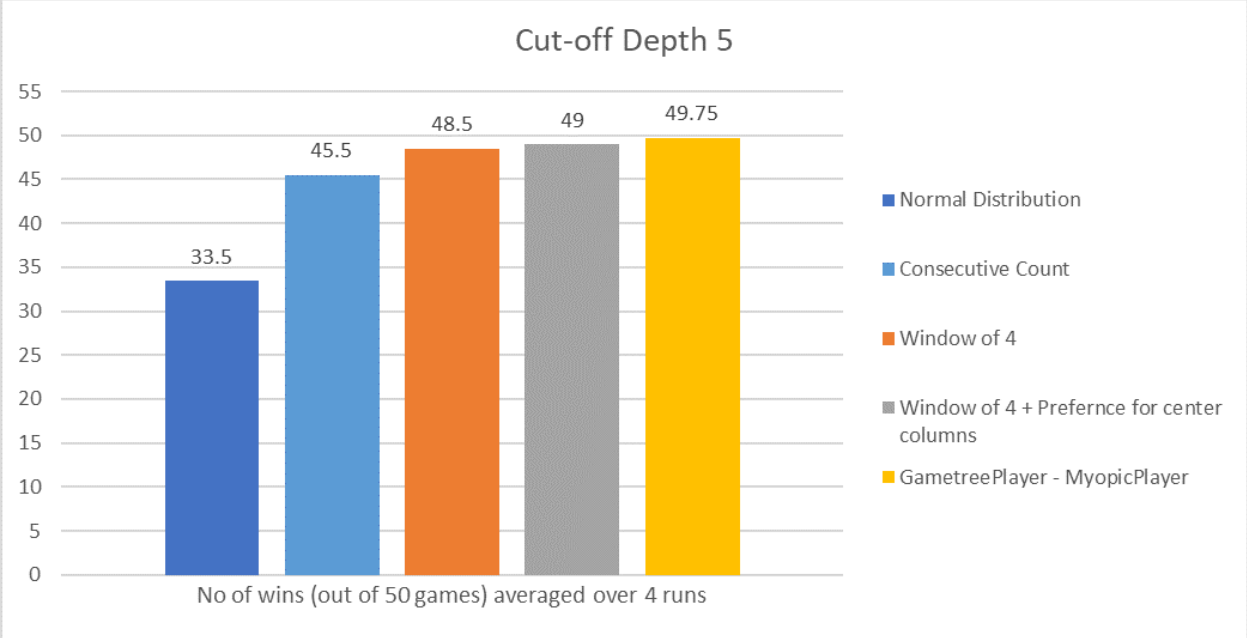
Cut-off Depth 5
55
50
45
40
35
30
25
20
15
10
5
0
33.5
45.5
48.5
49
49.75
No of wins (out of 50 games) averaged over 4 runs
Normal Distribution
Consecutive Count
Window of 4
Window of 4 + Prefernce for center columns
GametreePlayer - MyopicPlayer

Fig. 3a

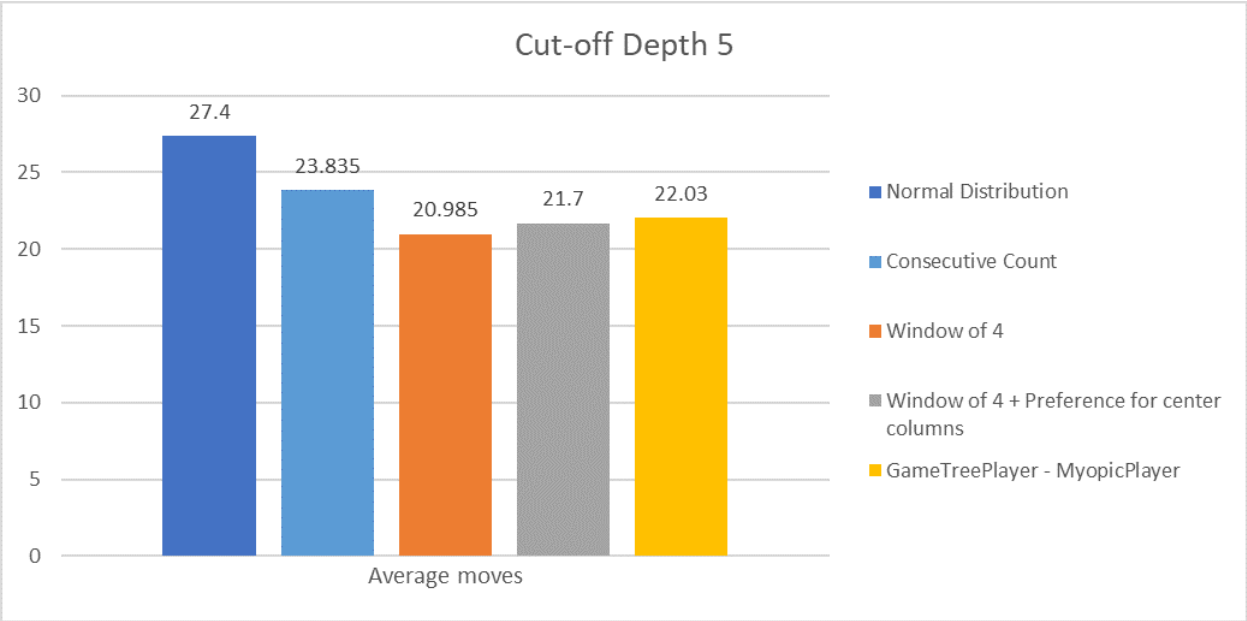

Fig. 3b

Alpha - Beta pruning

Recursive Calls

Alpha - Beta pruning leads to a 6-fold reduction in the number of recursive calls in the minimax algorithm with depth 5. In the case of depth 3, it leads to a 2.2-fold decrease (Fig. 4a and 4b).

Time

Alpha - Beta pruning leads to a 5.4-fold decrease in total runtime in the case of depth five and a 2.7-fold reduction when it comes to depth 3 (Fig. 5a and 5b).

Move Ordering

Move ordering has been implemented such that the middle columns are checked first. This improves performance by a significant margin.

Recursive Calls

Move ordering led to an improvement of 2.3-fold decrease of recursive calls over and above alpha-beta pruning, which is 14.2 times better than vanilla MiniMax for depth 5. For a depth of 3, it decreases by a factor of 3.3 (Fig. 4a and 4b).

Time

Move ordering led to 2.6 times better performance than alpha-beta pruning, translating to a 14 times better performing algorithm than vanilla MiniMax. For a depth of 3, it performs 8 times better than vanilla MiniMax (Fig. 5a and 5b).

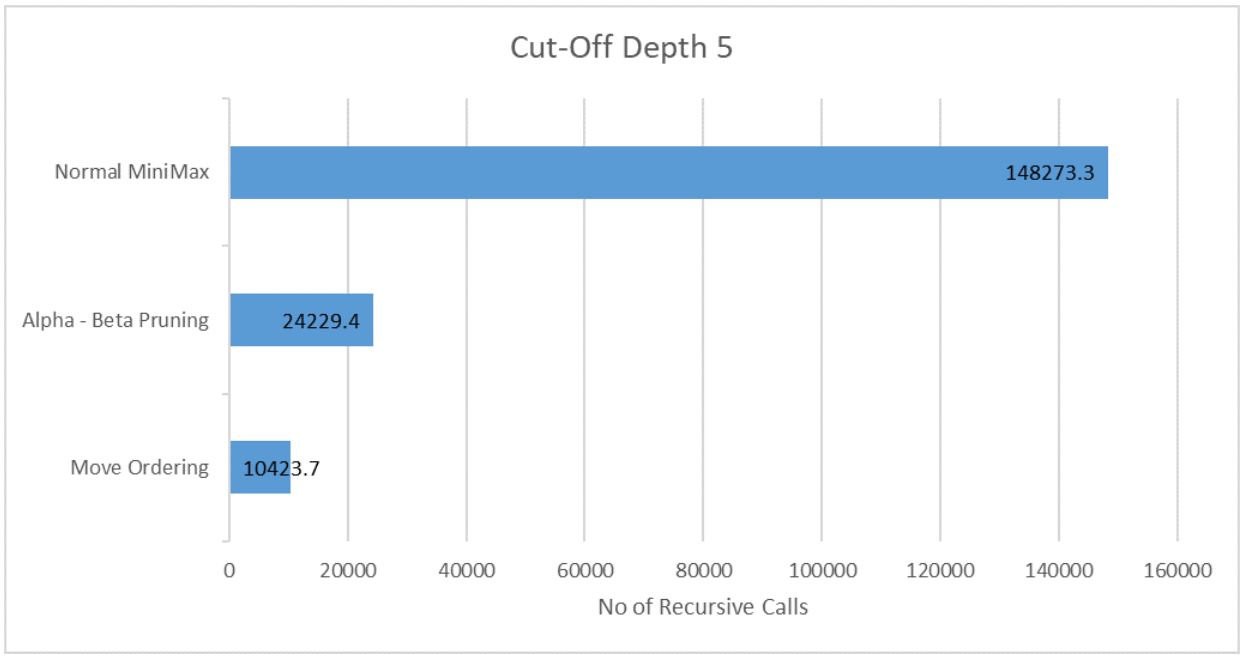

Fig. 4a

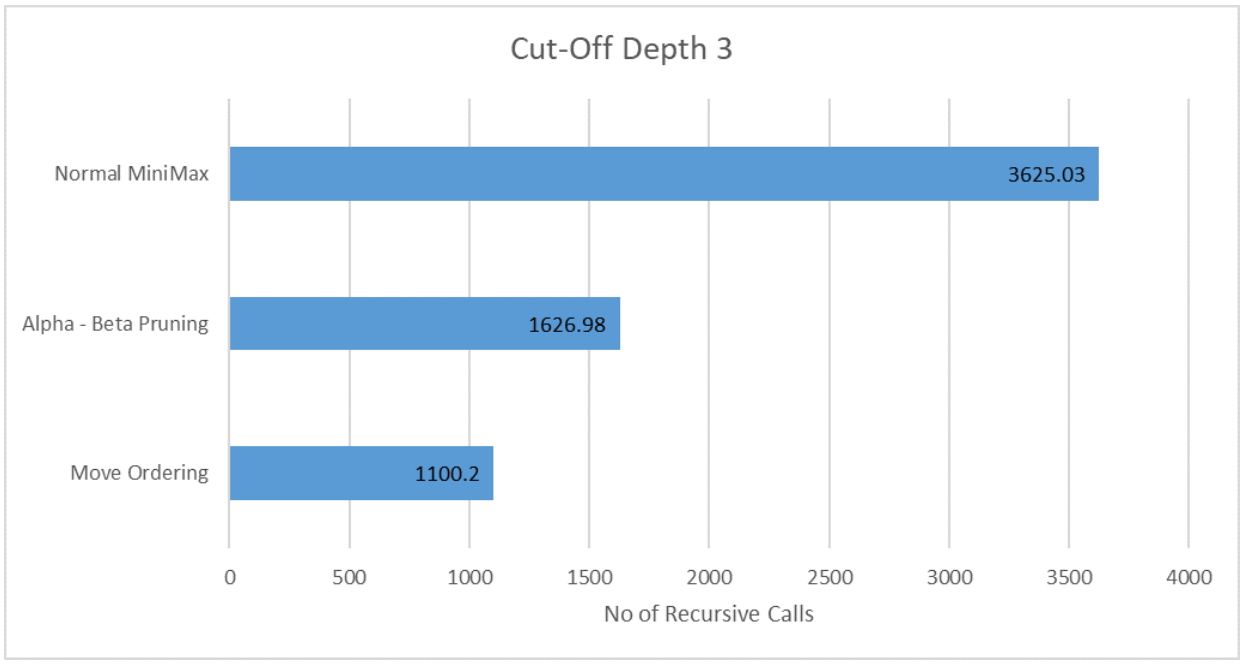
Cut-Off Depth 3
Normal MiniMax
3625.03
Alpha - Beta Pruning
1626.98
Move Ordering
1100.2
0
500
1000
1500
2000
2500
3000
3500
4000
No of Recursive Calls

Fig. 4b

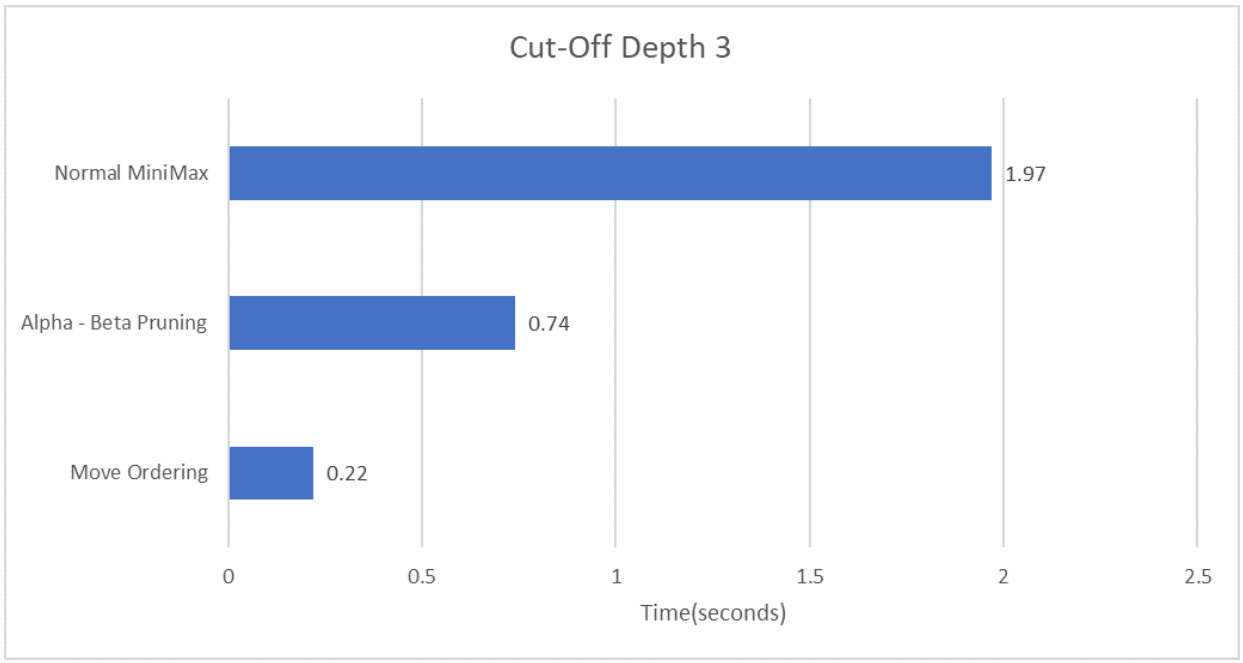
Cut-Off Depth 3
Normal MiniMax
1.97
Alpha - Beta Pruning
0.74
Move Ordering
0.22
0
0.5
1
1.5
2
2.5
Time(seconds)

Fig. 5a

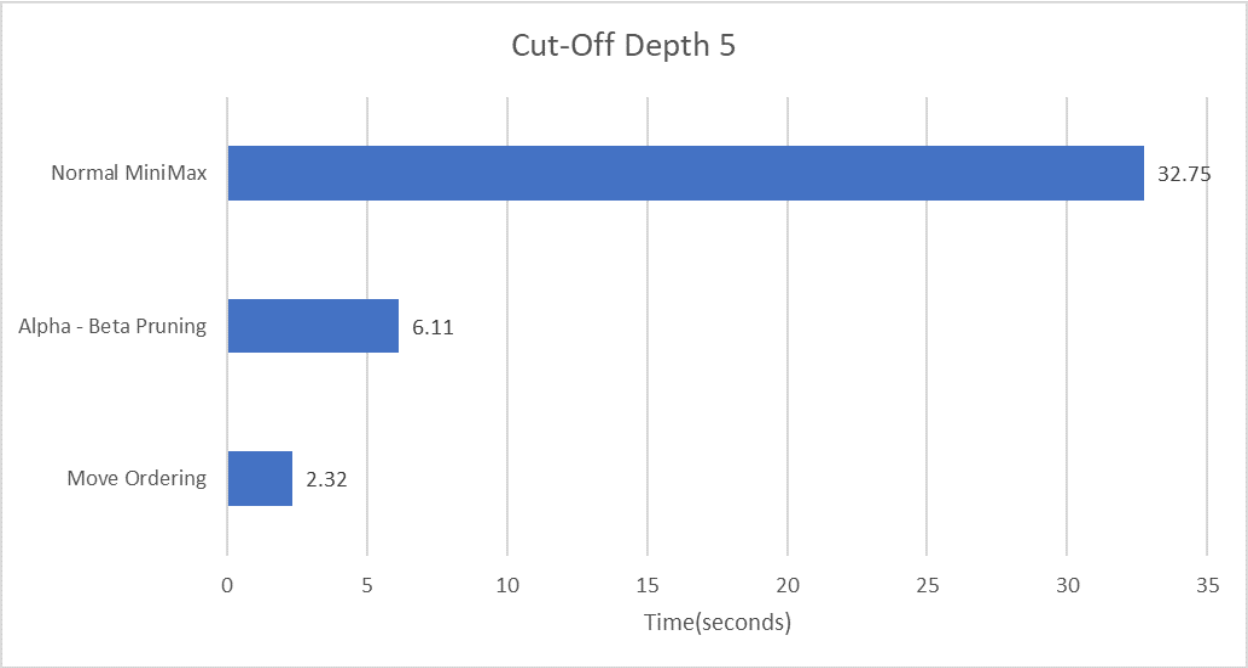

Fig. 5b

Conclusion

Increasing the cut-off depth to five consistently improves the win rate for all evaluation functions but has no significant impact on the average number of moves. The evaluation function, which considers the windows of both GamtreePlayer and MyopicPlayer and uses their difference, gives the best winning performance for both depths 3 and 5. It almost always wins for a depth of 5. **Implementing move-ordering leads to the best performance in terms of the total time required for a game, and this difference is even more pronounced for higher depth values and leads to a multifold decrease in number of recursive calls.**