University of Tehran

Faculty of Engineering

School of ECE

AI

CA2 - report sheet

Niloufar Mortazavi

SID : 220701096

Game

I implemented connect4 game with minimax algorithm, the cpu as the minimax player and the human player as a random player.

To optimize the performance of the minimax algorithm, I added a prune attribute, allowing alpha-beta pruning to be toggled on or off. When prune is True, the algorithm skips unnecessary branches, reducing nodes visited and computation time. With prune set to False, the algorithm evaluates all moves, which is more thorough but slower. This setup allows easy comparison of the algorithm's performance with and without pruning.

To fully understand the algorithm, the code was executed at various depths without alpha-beta pruning, and the average time, win probability, and the number of nodes visited were recorded for each depth.

"Pruning Disabled"

Depth	User Wins	CPU Wins	Ties	Total Time (s)	Nodes Visited
1	24	74	2	21.92	7544
2	2	97	1	79.52	54214
3	14	85	1	1379.60	74200

Average Nodes Visited:

- Depth 1: Average nodes visited was 75.44
- Depth 2: Average nodes visited was approximately 542.14
- Depth 3: Average nodes visited was around 742.

Average Runtime:

- Depth 1: Average runtime ≈ 0.21 seconds.
- Depth 2: Average runtime ≈ 0.79 seconds.
- Depth 3: Average runtime ≈ 13.80 seconds.

Average Winning Rates:

- Depth 1: User Winning Chance: 24%, CPU Winning Chance: 74%
- Depth 2: User Winning Chance: 2%, CPU Winning Chance: 97%
- Depth 3: User Winning Chance: 14%, CPU Winning Chance: 85%

"Pruning Enabled"

Depth	User Wins	CPU Wins	Ties	Total Time (s)	Nodes Visited
1	22	74	4	15.01	7469
2	1	93	6	45.23	31439
3	2	96	2	189.29	134638

Average Nodes Visited:

- Depth 1: Average nodes visited per run was 74.69
- Depth 2: Average nodes visited per run was approximately 314.39
- Depth 3: Average nodes visited per run was around 1346.38

Average Runtime:

- Depth 1: Average runtime ≈ 0.15 seconds.
- Depth 2: Average runtime ≈ 0.45seconds.
- Depth 3: Average runtime ≈ 1.89 seconds.

Average Winning Rates:

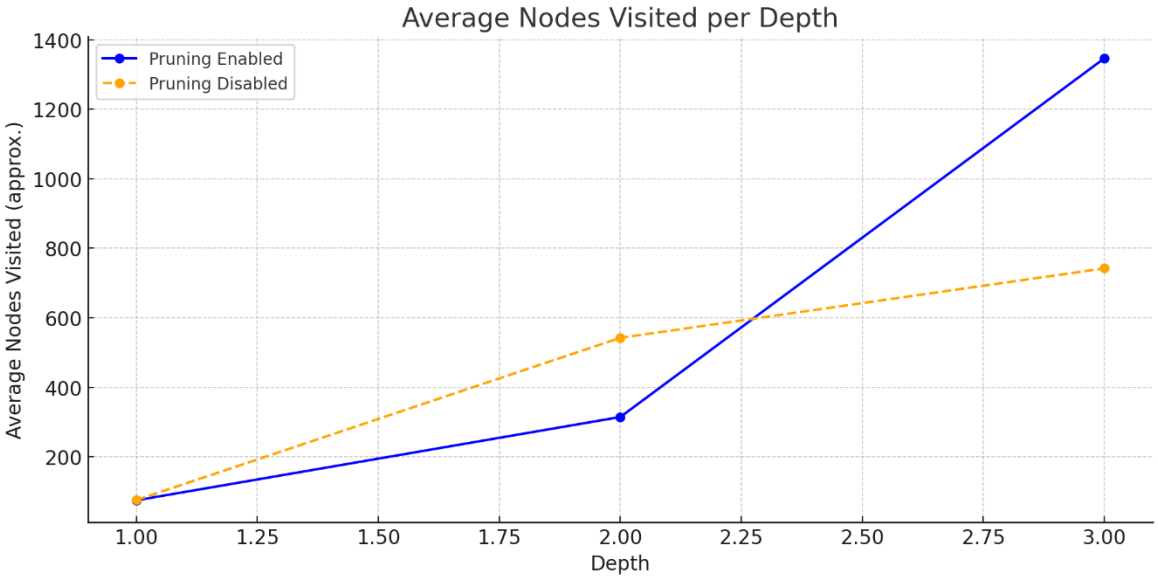
- Depth 1: User Winning Chance: 22%, CPU Winning Chance: 74%
- Depth 2: User Winning Chance: 1%, CPU Winning Chance: 93%
- Depth 3: User Winning Chance: 2%, CPU Winning Chance: 96%

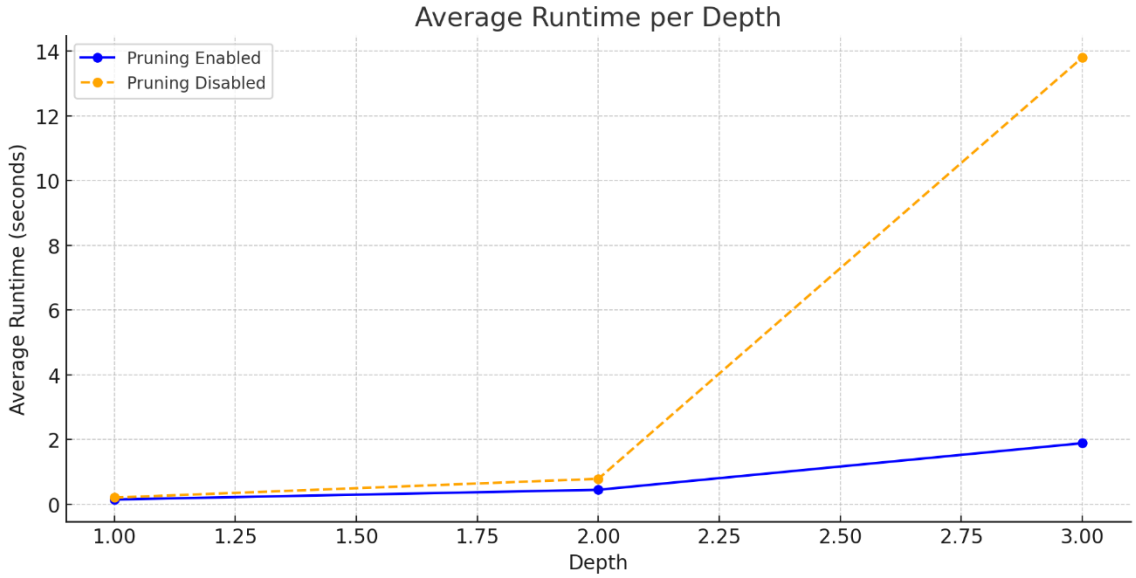
With pruning enabled, the minimax algorithm shows a significant reduction in the average nodes visited and runtime across all depths compared to when pruning is disabled. This efficiency gain does not impact accuracy, as the CPU's winning chances remain high in both cases, though the enabled pruning sometimes slightly increases the user's winning rate. Overall, pruning improves the algorithm's efficiency without sacrificing performance.

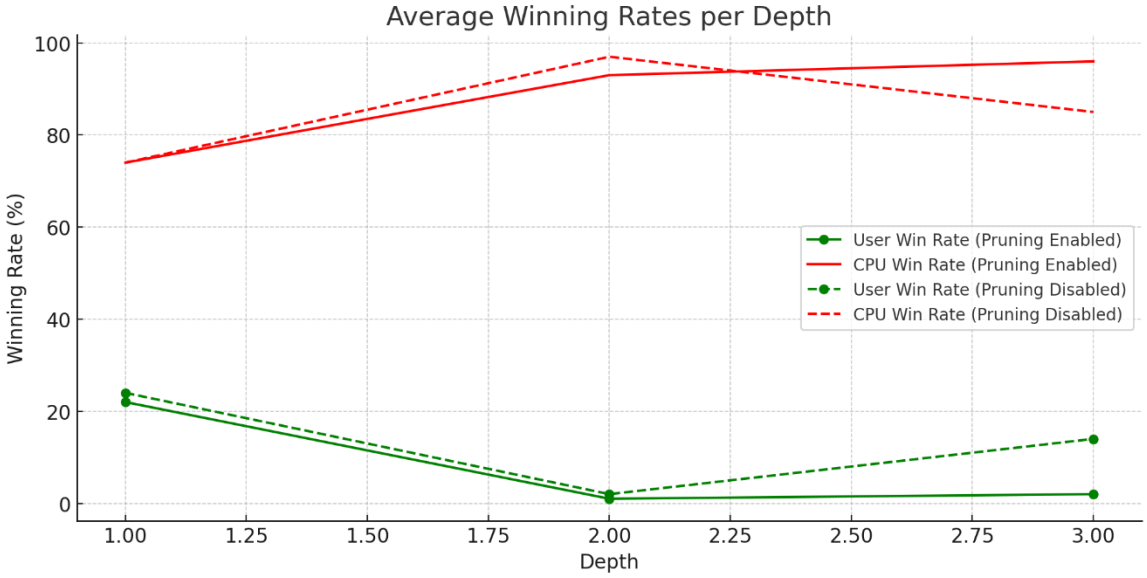
Q1- Is there any relationship between the depth of the algorithm and the parameters above?

As the depth increases, both the average runtime and the number of nodes visited tend to increase significantly, due to the exponential growth in possible game states explored at each level. This trend is more pronounced when pruning is disabled, as the algorithm explores more nodes without cutting off branches.

Additionally, the winning chances for the CPU generally increase with depth, suggesting that deeper searches allow the CPU to make more strategically optimal moves. However, this increased CPU advantage comes at the cost of higher computational effort, particularly in terms of time and nodes visited.


Average Nodes Visited per Depth
Pruning Enabled
Pruning Disabled
Average Nodes Visited (approx.)
1400
1200
1000
800
600
400
200
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Depth


Average Runtime per Depth
Pruning Enabled
Pruning Disabled
Average Runtime (seconds)
14
12
10
8
6
4
2
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Depth


Average Winning Rates per Depth
Winning Rate (%)
100
80
60
40
20
0
User Win Rate (Pruning Enabled)
CPU Win Rate (Pruning Enabled)
User Win Rate (Pruning Disabled)
CPU Win Rate (Pruning Disabled)
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Depth

Q2- Is it possible to choose the order of exploring each node's children to achieve maximum pruning?

Yes, it is possible to order the exploration of child nodes in a way that maximizes pruning. In the minimax algorithm with alpha-beta pruning, we can prioritize nodes that are more likely to produce the best move early on.

To achieve this ordering, we can evaluate each child node with a simple heuristic before exploring it fully in the minimax search. For example, in a Connect4 game, moves in the center column are generally stronger, as they provide more potential connections. Therefore, by exploring moves in the center columns first and then moving outward, we increase the likelihood of finding optimal moves early.

Q3 – Branching factor's changes

In the context of the minimax algorithm, **branching factor** refers to the number of possible moves (child nodes) that minimax explores at each level of the game tree. In Connect4, the branching factor for minimax depends on the number of columns that are open for a piece to be placed.

At the start of the game, the branching factor is high—up to seven—since all columns are available. As the game progresses and some columns fill up, the branching factor in minimax decreases, as there are fewer valid moves available in each position. This means that minimax has fewer branches to evaluate as the board fills up, making the later stages of the game less computationally intensive.

Early in the game, however, the high branching factor requires minimax to explore a larger number of possible moves, increasing the algorithm's computational load. Therefore, the branching factor in minimax decreases as the game advances, balancing the search effort over the course of the game.

Q4- Why does the algorithm become faster during pruning without losing its accuracy?

When the minimax algorithm uses **pruning**, it becomes faster because it skips over moves that won't affect the final decision. Pruning works by setting limits (alpha and beta) for the best scores found so far. If a branch (a possible move) is found that won't lead to a better outcome than what's already been found, the algorithm stops looking down that path. This way, it doesn't waste time on moves that won't change the result. By ignoring these unnecessary branches, pruning lets the algorithm make the same best decision but in less time.

Q5- Why isn't minimax the optimal method when the opponent acts randomly (like in this project)? What algorithm could replace it?

When the opponent plays randomly, using the minimax algorithm isn’t the most optimal approach. Minimax is designed based on the assumption that the opponent will make the best possible moves to win. However, if the opponent's moves are random, minimax ends up overestimating the complexity of the game, evaluating many scenarios that don’t actually reflect the randomness of the opponent.

A better alternative in this case would be algorithms like Monte Carlo Tree Search (MCTS). MCTS simulates many random games to explore potential moves, gathering statistics on win rates based on random plays rather than assuming optimal opponent moves. This makes MCTS more adaptable to situations where the opponent plays unpredictably, providing a more practical approach for games against a random-playing opponent.