CSE 3111 / CSE 3213

ARTIFICIAL INTELLIGENCE

FALL 2023

Programming Assignments Report

Furkan Özkaya – 200316060

Furkan Baytak – 210316033

Sıla Naz Aslan – 210316072

Submission Date: 3 January 2024

1 Development Environment

Python 3.12.1, Windows 11, Visual Studio Code IPYNB

2 Problem Formulation

State Specification: Status represents the current arrangement of queens on the chessboard. Each queen occupies a unique row and column on the board. It is represented as a tuple/list where each index corresponds to a row and the value at that index represents the column where the queen is placed.

Initial State: The initial state is either user-defined (entered manually by the user) or randomly generated. It represents the initial placement of queens on the board.

Possible Actions: Possible actions define how a queen can be moved within its column and how it can change its order. Actions are represented as tuples where the first element is the index of the queen to be moved and the second element is the new row for that queen.

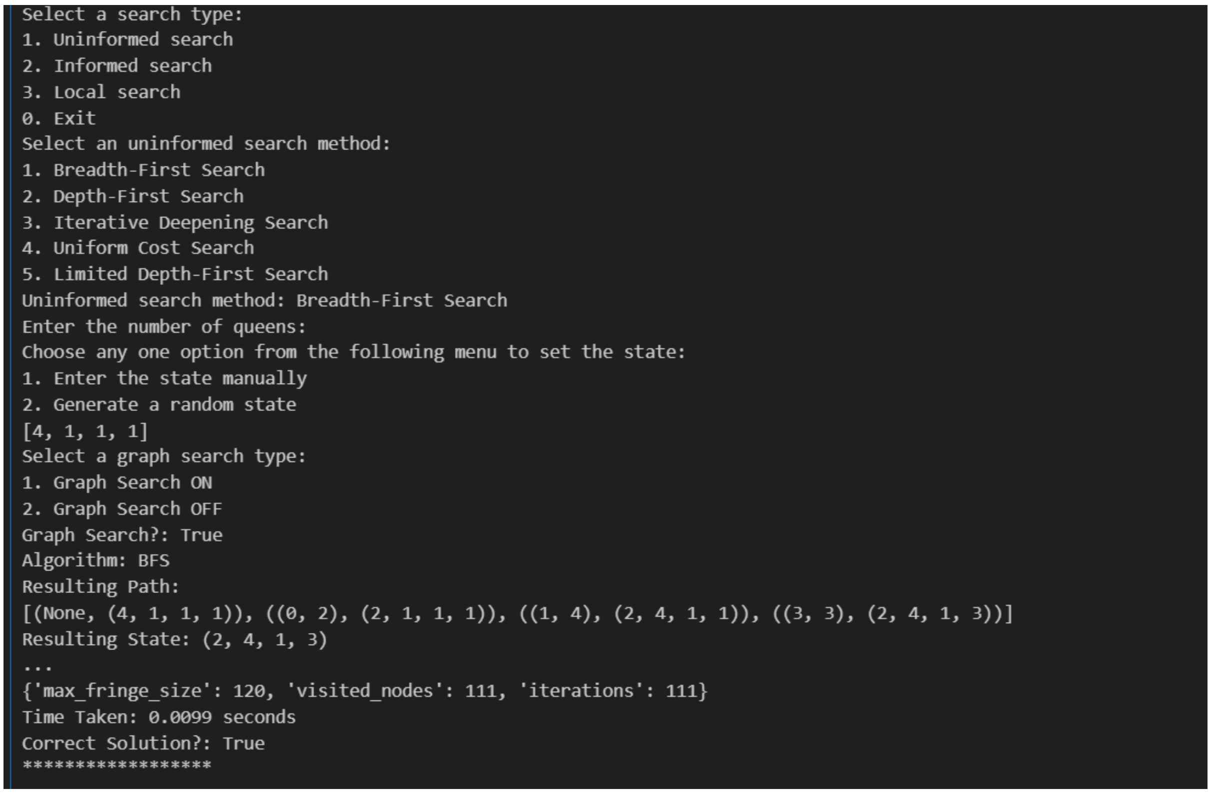
Transition Model: The transition model describes how actions change state. It creates a new state by moving a queen to a different row within its column.

Goal Test: The goal test checks whether the current situation represents a solution where the two queens do not threaten each other. This condition is met when there is no attacking queen pair on the board.

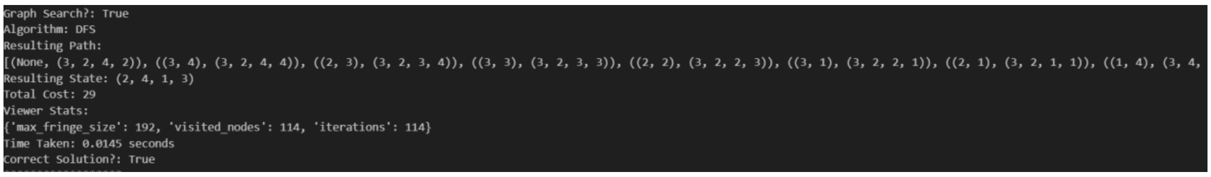
Path Cost: Since this problem does not require a cost to reach the goal state, the path cost is not explicitly defined in this formulation. However, the heuristic function calculates a value representing the number of attacking queen pairs, which helps informed search algorithms to reach the solution faster.

In a nutshell, the problem aims to find a configuration of N queens on an N×N chessboard where no two queens threaten each other, and uses various search algorithms to navigate the state space to reach a valid solution state.

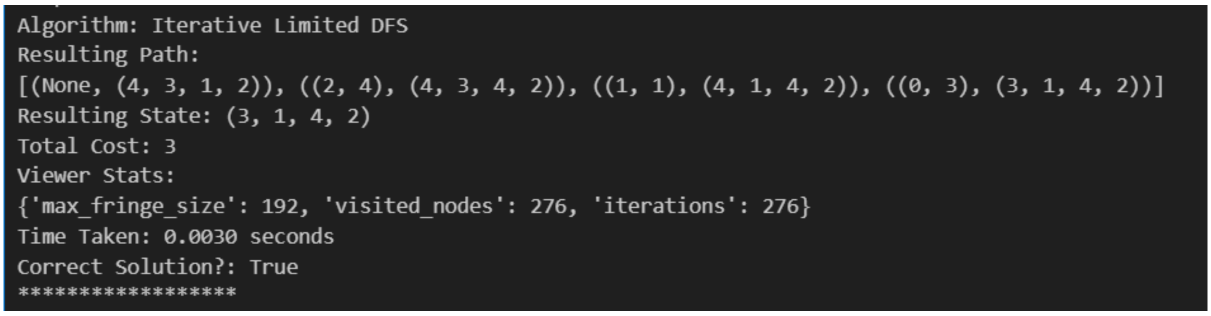
3 Results



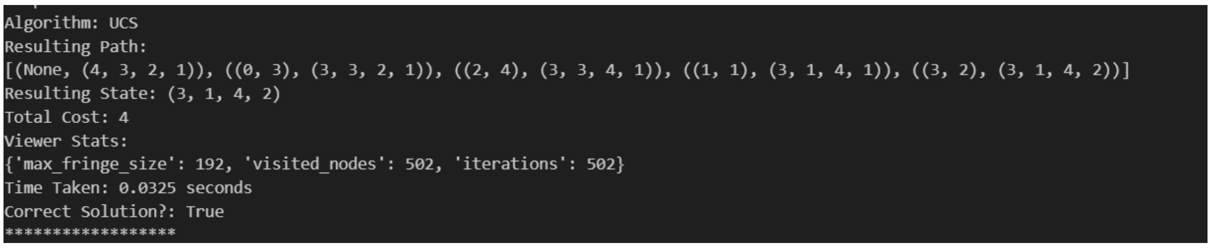
```
Select a search type:
1. Uninformed search
2. Informed search
3. Local search
0. Exit
Select an uninformed search method:
1. Breadth-First Search
2. Depth-First Search
3. Iterative Deepening Search
4. Uniform Cost Search
5. Limited Depth-First Search
Uninformed search method: Breadth-First Search
Enter the number of queens:
Choose any one option from the following menu to set the state:
1. Enter the state manually
2. Generate a random state
[4, 1, 1, 1]
Select a graph search type:
1. Graph Search ON
2. Graph Search OFF
Graph Search?: True
Algorithm: BFS
Resulting Path:
[(None, (4, 1, 1, 1)), ((0, 2), (2, 1, 1, 1)), ((1, 4), (2, 4, 1, 1)), ((3, 3), (2, 4, 1, 3))]
Resulting State: (2, 4, 1, 3)
...
{'max_fringe_size': 120, 'visited_nodes': 111, 'iterations': 111}
Time Taken: 0.0099 seconds
Correct Solution?: True
******************
```



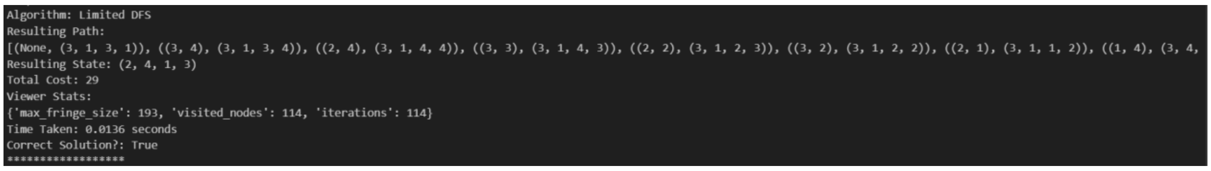
```
Graph Search?: True
Algorithm: DFS
Resulting Path:
[(None, (3, 2, 4, 2)), ((3, 4), (3, 2, 4, 4)), ((2, 3), (3, 2, 3, 4)), ((3, 3), (3, 2, 3, 3)), ((2, 2), (3, 2, 2, 3)), ((3, 1), (3, 2, 2, 1)), ((2, 1), (3, 2, 1, 1)), ((1, 4), (3, 4,
Resulting State: (2, 4, 1, 3)
Total Cost: 29
Viewer Stats:
{'max_fringe_size': 192, 'visited_nodes': 114, 'iterations': 114}
Time Taken: 0.0145 seconds
Correct Solution?: True
```



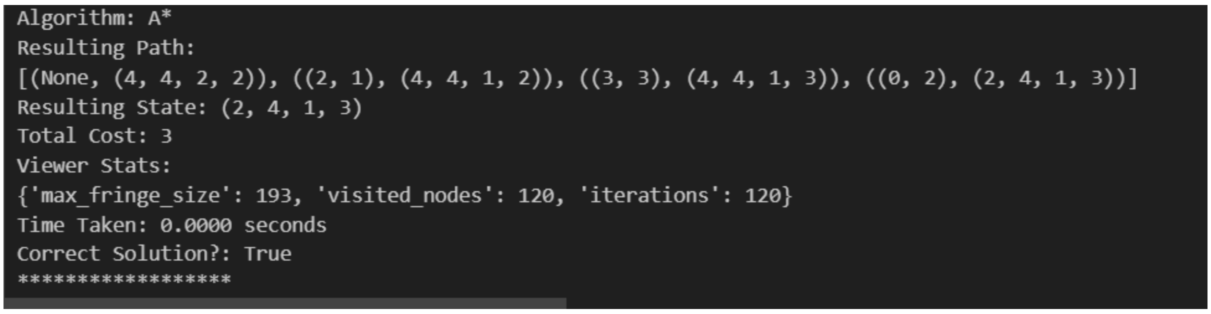
```
Algorithm: Iterative Limited DFS
Resulting Path:
[(None, (4, 3, 1, 2)), ((2, 4), (4, 3, 4, 2)), ((1, 1), (4, 1, 4, 2)), ((0, 3), (3, 1, 4, 2))]
Resulting State: (3, 1, 4, 2)
Total Cost: 3
Viewer Stats:
{'max_fringe_size': 192, 'visited_nodes': 276, 'iterations': 276}
Time Taken: 0.0030 seconds
Correct Solution?: True
******************
```



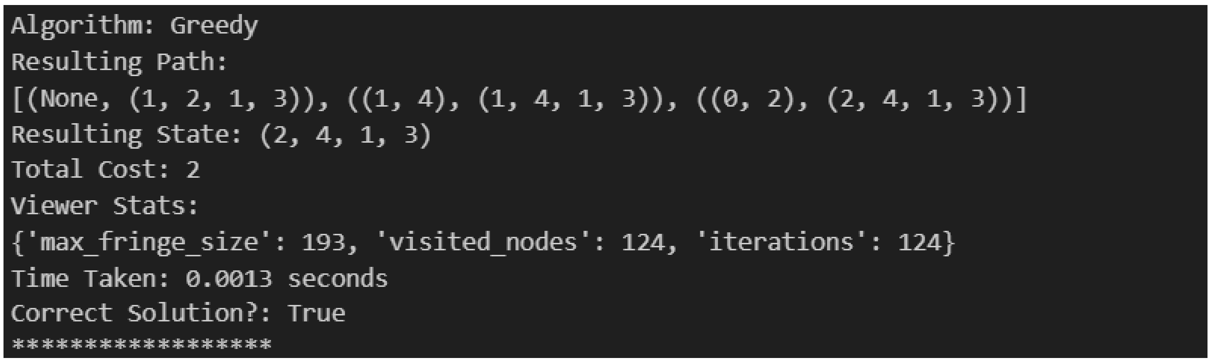
```
Algorithm: UCS
Resulting Path:
[(None, (4, 3, 2, 1)), ((0, 3), (3, 3, 2, 1)), ((2, 4), (3, 3, 4, 1)), ((1, 1), (3, 1, 4, 1)), ((3, 2), (3, 1, 4, 2))]
Resulting State: (3, 1, 4, 2)
Total Cost: 4
Viewer Stats:
{'max_fringe_size': 192, 'visited_nodes': 502, 'iterations': 502}
Time Taken: 0.0325 seconds
Correct Solution?: True
******************
```



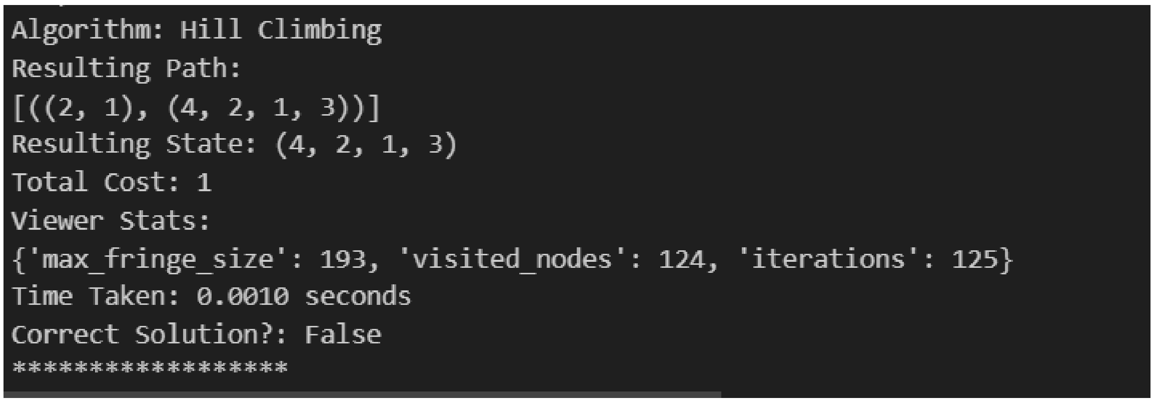
```
Algorithm: Limited DFS
Resulting Path:
[(None, (3, 1, 3, 1)), ((3, 4), (3, 1, 3, 4)), ((2, 4), (3, 1, 4, 4)), ((3, 3), (3, 1, 4, 3)), ((2, 2), (3, 1, 2, 3)), ((3, 2), (3, 1, 2, 2)), ((2, 1), (3, 1, 1, 2)), ((1, 4), (3, 4,
Resulting State: (2, 4, 1, 3)
Total Cost: 29
Viewer Stats:
{'max_fringe_size': 193, 'visited_nodes': 114, 'iterations': 114}
Time Taken: 0.0136 seconds
Correct Solution?: True
******************
```



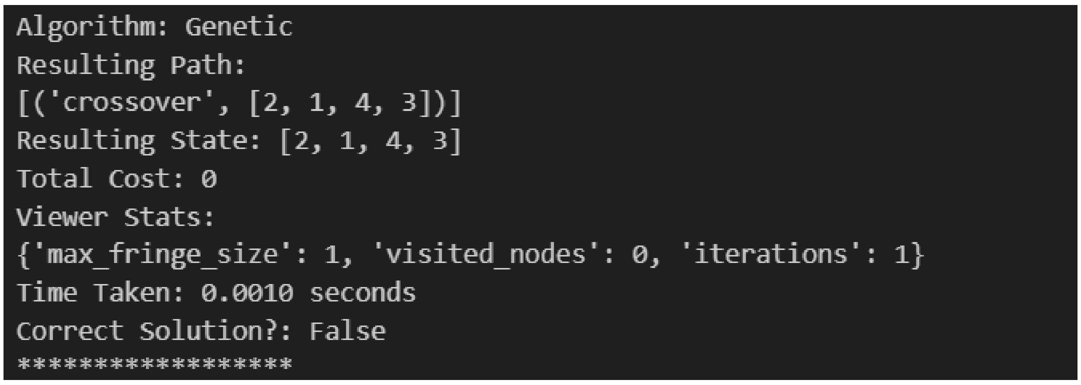
```
Algorithm: A*
Resulting Path:
[(None, (4, 4, 2, 2)), ((2, 1), (4, 4, 1, 2)), ((3, 3), (4, 4, 1, 3)), ((0, 2), (2, 4, 1, 3))]
Resulting State: (2, 4, 1, 3)
Total Cost: 3
Viewer Stats:
{'max_fringe_size': 193, 'visited_nodes': 120, 'iterations': 120}
Time Taken: 0.0000 seconds
Correct Solution?: True
******************
```



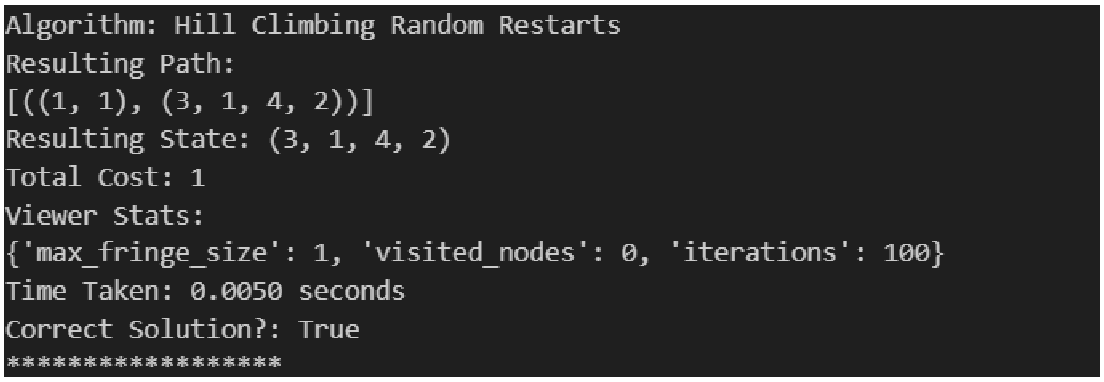
```
Algorithm: Greedy
Resulting Path:
[(None, (1, 2, 1, 3)), ((1, 4), (1, 4, 1, 3)), ((0, 2), (2, 4, 1, 3))]
Resulting State: (2, 4, 1, 3)
Total Cost: 2
Viewer Stats:
{'max_fringe_size': 193, 'visited_nodes': 124, 'iterations': 124}
Time Taken: 0.0013 seconds
Correct Solution?: True
******************
```



```
Algorithm: Hill Climbing
Resulting Path:
[((2, 1), (4, 2, 1, 3))]
Resulting State: (4, 2, 1, 3)
Total Cost: 1
Viewer Stats:
{'max_fringe_size': 193, 'visited_nodes': 124, 'iterations': 125}
Time Taken: 0.0010 seconds
Correct Solution?: False
******************
```



```
Algorithm: Genetic
Resulting Path:
[('crossover', [2, 1, 4, 3])]
Resulting State: [2, 1, 4, 3]
Total Cost: 0
Viewer Stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 1}
Time Taken: 0.0010 seconds
Correct Solution?: False
******************
```



```
Algorithm: Hill Climbing Random Restarts
Resulting Path:
[((1, 1), (3, 1, 4, 2))]
Resulting State: (3, 1, 4, 2)
Total Cost: 1
Viewer Stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 100}
Time Taken: 0.0050 seconds
Correct Solution?: True
******************
```

4 Discussion

Completeness:

Completeness refers to whether an algorithm is guaranteed to find a solution. Algorithms that search extended state spaces are usually exact. For example, BFS (Breadth-First Search) and UCS (Uniform Cost Search) are exact. In contrast, DFS (Depth-First Search) is incomplete as it can get stuck in infinite loops in a finite state space.

Optimality:

Optimality refers to whether an algorithm is guaranteed to find the best solution. Some algorithms are guaranteed to find the best solution, while others may find a solution but not necessarily the best solution. For example, optimal search algorithms such as A and UCS guarantee finding the best solution, while DFS and Greedy do not.*

Time and Space Complexity:

Each algorithm has different time and space complexity. Algorithms that expand the state space extensively, such as BFS, require more space. Meanwhile, DFS uses less space but can perform poorly in terms of time complexity. Informed search algorithms like A typically deliver more efficient results.*

The Effect of Depth Limit in Depth Limited Search:

Depth-limited search aims to improve DFS by restricting its depth. However, setting a depth limit risks missing the solution. A depth-limited search that does not have enough depth may not find the optimal solution. Increasing the depth limit can improve performance but can also increase time and space utilization.Comparison of Local Search Algorithms to

Traditional Search Algorithms:

Local search algorithms usually do not guarantee global optimality, but they can be effective in large state spaces. In particular, they can be successful in finding solutions quickly from a single starting point in large state spaces. However, it is very important to note that these algorithms can get stuck at local minima or have limitations in improving valid solutions.

These observations depend on problem domains, state space dimensions and specific problem characteristics. For example, if a succinct solution is sought, exhaustive search algorithms such as BFS or UCS may be preferable. However, if there are time or space constraints, deliberate search algorithms such as A may be more appropriate.*

The nature of the problem, performance requirements and available resources should be considered when choosing the most appropriate algorithm for each problem and situation.