FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMANT

2021-2022 FALL

ARTIFICIAL INTELLIGENCE

ASSIGNMENT 2-3 REPORT

02.01.2022

180315033

HÜSNA İBİŞ

Development Environment:

Python version is 'Python 3.7.4'. The OS is Windows. I used Anaconda, Spyder. Python Anaconda is an integrated python distribution system for developers who want to develop various similar scientific applications such as data science, analysis, and machine learning using Python. It includes developer interfaces such as "Spyder" , "Jupyter Notebook" and "JupyterLab". CPU has the properties "Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz".

Problem Formulation:

In this program, states are string format. States indicate the position of the queens on the board. The elements of the String tell which row the queen is in, the index of the element tells which queen it is (first queen, second queen...) and which column it is in. The program is designed to have a queen in each column. Since this is the basis, we don't need to look at the columns while counting the attacking pairs. We look at rows and diagonals. Since we have an N*N board, its elements must not be greater than N and less than 1 for the given state to be valid. Also, the state string must be N long.

When calling the search algorithms, the program gives us two options for creating the initial state. We can create the state manually, or the program can randomly create a valid state for us. Initial state determines from which state the search algorithm will start the search.

Starting from the initial state, the states to go are determined by the actions. The Actions array contains the actions that can be performed from that state. For example, if we add the row value of the 1st queen with 1, so that the 1st queen can go 1 step forward, we will move the queen 1 forward. We assign these actions to an array named "actions" within the "actions" function. So the actions array [(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1), (-1,0, 0,0),(0,-1,0,0),(0,0,-1,0),(0,0,0,-1)]. Since the SimpleAI library does not accept array format, I converted the elements of the array to tuple format.

In the result function, when we convert the given action and the state to the appropriate formats and add them, we move the related queen forward or backward. The result function returns the new state resulting from this action. The function validates the new state with the "is_valid" function. if the result is false that action will not occur.

After the action is performed, a goal test is applied to the new state. (if new state is valid). For the state to be a goal state, the number of attacking pairs must be 0. Within the is_goal function, we perform the goal test with the help of the count_attacking_pairs function that we created in the 1st assignment. If it returns true, the algorithm terminates. The algorithm tells us what goal state is and which path used to found goal state.

Results:

State = 4311, N=4:

Depth First Search;

Time = 764 saniye, Max fringe size = 154, Visited nodes = 131, Iterations = 131, Goal state = 2413, Viewer = WebViewer, Cost = 59, Path = (Very long. Because cost is 59)

Time = 2 saniye, Max fringe size = 154, Visited nodes = 131, Iterations = 131, Goal state = 2413, Viewer = ConsoleViewer, Cost = 59

Breadth First Search;

Time = 2 saniye, Max fringe size = 51, Visited nodes = 93, Iterations = 93, Goal state = 2413, Viewer = ConsoleViewer, Cost = 5, Path = [(None, '4311'), ((-1, 0, 0, 0), '3311'), ((-1, 0, 0, 0), '2311'), ((0, 1, 0, 0), '2411'), ((0, 0, 0, 1), '2412'), ((0, 0, 0, 1), '2413')]

Time = 275 saniye, Max fringe size = 51, Visited nodes = 93, Iterations = 93, Goal state = 2413, Viewer = WebViewer, Cost = 5, Path = [(None, '4311'), ((-1, 0, 0, 0), '3311'), ((-1, 0, 0, 0), '2311'), ((0, 1, 0, 0), '2411'), ((0, 0, 0, 1), '2412'), ((0, 0, 0, 1), '2413')]

Limited Depth First;

Time = 3.9 saniye, Max fringe size =53 , Visited nodes = 39, Iterations = 39, Goal state = 3142, Viewer = ConsoleViewer, Cost = 19, Depth limit=20

Time = 1.4 saniye, Max fringe size = 36, Visited nodes = 175, Iterations = 175, Goal state = 2413, Viewer = ConsoleViewer, Cost = 7, Depth limit = 10

Time = 1.2 saniye, Max fringe size = 19, Visited nodes = 44, Iterations = 44, Goal state = 2413, Viewer = ConsoleViewer, Cost = 5, Depth limit = 5

Time = 1.4 saniye, Max fringe size = 36, Visited nodes = 175, Iterations = 175, Goal state = 2413, Viewer = ConsoleViewer, Cost = 7, Depth limit = 10

Iterative Limited Depth First;

Time = 1.4 saniye, Max fringe size = 19, Visited nodes = 170, Iterations = 170, Goal state = 2413, Viewer = ConsoleViewer, Cost = 5

Uniform Cost;

Time = 1.4 saniye, Max fringe size = 65, Visited nodes = 109, Iterations = 109, Goal state = 2413, Viewer = ConsoleViewer, Cost = 5

Greedy;

Time = 0.6 saniye, Max fringe size = 31, Visited nodes = 11, Iterations = 11, Goal state = 2413, Viewer = ConsoleViewer, Cost = 7

A;*

Time = 1.2 saniye, Max fringe size = 54, Visited nodes = 29, Iterations = 29, Goal state = 2413, Viewer = ConsoleViewer, Cost = 5

State = 13154, N=5:

Depth First Search;

Time = 2 saniye, Max fringe size = 1217, Visited nodes = 328, Iterations = 328, Goal state = 35241, Viewer = ConsoleViewer, Cost = 327

Breadth First Search;

Time = 2 saniye, Max fringe size = 108, Visited nodes = 66, Iterations = 66, Goal state = 14253, Viewer = ConsoleViewer, Cost = 3

Limited Depth First;

Time = 1.6 saniye, Max fringe size =90 , Visited nodes = 1144, Iterations = 1144, Goal state = 14253, Viewer = ConsoleViewer, Cost = 17, Depth limit=20

Time = 1 saniye, Max fringe size = 50, Visited nodes = 508, Iterations = 508, Goal state = 14253, Viewer = ConsoleViewer, Cost = 7, Depth limit = 10

Time = 1.2 saniye, Max fringe size = 25, Visited nodes = 61, Iterations = 61, Goal state = 14253, Viewer = ConsoleViewer, Cost = 5, Depth limit = 5

Iterative Limited Depth First;

Time = 1.4 saniye, Max fringe size = 17, Visited nodes = 61, Iterations = 61, Goal state = 14253, Viewer = ConsoleViewer, Cost = 3

Uniform Cost;

Time = 1.3 saniye, Max fringe size = 133, Visited nodes = 79, Iterations = 79, Goal state = 35241, Viewer = ConsoleViewer, Cost = 3

Greedy;

Time = 1 saniye, Max fringe size = 16, Visited nodes = 4, Iterations = 4, Goal state = 14253, Viewer = ConsoleViewer, Cost = 3

A;*

Time = 1.2 saniye, Max fringe size = 21, Visited nodes = 5, Iterations = 5, Goal state = 14253, Viewer = ConsoleViewer, Cost = 3

Discussion:

I ran the search algorithms using Console Viewer and Web Viewer. When I used webviewer, it took so long to get results. But the webviewer presents the results more clearly. Since I want to get results in a short time, I run the algorithms mostly using console viewer. When I used the console viewer, the search algorithm ended within seconds and gave me the goal state in a short time.

Breadth First Search, Uniform Cost Search, A* and Iterative Limited Depth First search algorithms are complete. So it always gives us the least cost solution. When I ran the program, these algorithms returned the solution with a cost of 5 for state 4311. But other algorithms returned solutions with high cost. Therefore, the Greedy, Depth First, and Limited Depth First search algorithms are not complete. The Greedy search algorithm returned the least cost solution for state 13154, but did not return the least cost solution for state 4311. Therefore, we cannot say the Greedy search algorithm complete.

Breadth First Search, Uniform Cost Search, A* and Iterative Limited Depth First search algorithms are optimal. A* graph-search is optimal if heuristic is consistent. When we reduce the depth limit too much in Limited Depth First search, the algorithm cannot find the goal state. Goal state is out of bounds. So, Limited Depth First search is not optimal search algorithm.

When we increased N, the running time of all algorithms increased. But Breadth First search and Depth First search algorithms worked much longer than others. We can say that the search

algorithms with the highest time complexity are Breadth First search and Depth First search. In the Limited Depth Search algorithm, we see that as the dept limit decreases, the fringe size and cost decrease. In other words, we can say that the space complexity and the dept limit change in direct proportion. When we compare maximum fringe size based on the information in the Result section, we actually learn about space complexity. The algorithm with the maximum fringe size is the Depth First search algorithm. So the algorithm with the highest space complexity is the Depth First search algorithm.