CSE 3111 / CSE 3213

ARTIFICIAL INTELLIGENCE

FALL 2023

Programming Assignments Report

Baran Çakmak Demir - 210315002

Submission Date: 5 January 2024

1 Development Environment

For these assignments, I have used Python 3.11 on a Windows 10 operating system. Also, for the integrated development environment (IDE) Visual Studio Code is chosen. Used machine for these assignments has Intel i7-11800h processor with 2.30 GHz base clock speed and max 4.6 GHz clock speed with 8 cores and 16 threads.

2 Problem Formulation

PA1:

In this assignment, we are wanted to arrange queens on a chessboard where they do not attack each other. For the testing, user can enter a state or it can be randomly generated.

PA2 and PA3:

In PA2 , we have learned how to apply uninformed and informed search algorithms and how to evaluate the performance of search algorithms.

In PA3, we have learned how to formulate a given problem as a search problem, how to solve it by applying the local search algorithms and how to evaluate the performance of search algorithms.

State Specification: It is used for how states are represented in the NQueens Problem.

Initial State: It is used for how user generate the initial state for this problem. There are 2 ways to generate initial state. One of them is user generating the initial state. Other one is initial state is randomly generated.

Possible Actions: Outline the possible actions for the problem.

Transition model: It is used for how to state changes when the action occurs.

Goal Test: It is used for determining the if a state is goal. If number of attacking pairs is 0 then it means, we reached goal.

Path Cost: It represent the cost of each action. It is considered as 1 for every action by default.

3 Results

PA1:

Part 2: Testing

Do not change this part. This is the test code.

```
# This is a test code. You can try with different N values and states.
problem = NQueens(7) #create NQueens instance
print(problem) #print the description of the problem
print(problem._count_attacking_pairs(problem.state)) #print the total number of attacking pairs in the board
```

[5] ✓ 26.1s

```
Representation of NQueens Problem:
N = 7
State = 1234567
Count attacking pairs: 21
```

Part 2: Testing

Do not change this part. This is the test code.

```
# This is a test code. You can try with different N values and states.
problem = NQueens(7) #create NQueens instance
print(problem) #print the description of the problem
print(problem._count_attacking_pairs(problem.state)) #print the total number of attacking pairs in the board
```

[6] ✓ 3.7s

```
Representation of NQueens Problem:
N = 7
State = 2574361
Count attacking pairs: 4
```

PA2 and PA3:

```
limited_depth_first
Resulting path:
[(None, '2323'), (('Move Queen 4 to row 4', 4, 4), '2324'), (('Move Queen 4 to row 4', 4, 4), '2324'), (('Move
Resulting state: 2413
Total costs: 5
Value: 6
Correct solution?: True
************************************************
************************************************
iterative_limited_depth_first
Resulting path:
[(None, '2323'), (('Move Queen 3 to row 1', 3, 1), '2313'), (('Move Queen 2 to row 4', 2, 4), '2413')]
Resulting state: 2413
Total costs: 2
Value: 6
Correct solution?: True
************************************************
************************************************
greedy
Resulting path:
[(None, '2323'), (('Move Queen 2 to row 4', 2, 4), '2423'), (('Move Queen 3 to row 1', 3, 1), '2413')]
Resulting state: 2413
Total costs: 2
Value: 6
Correct solution?: True
```

```
hill_climbing_random_restarts
Resulting path:
[(('Move Queen 4 to row 3', 4, 3), '2413')]
Resulting state: 2413
Total costs: 2
Value: 6
Correct solution?: True
************************************************
************************************************
genetic
Resulting path:
[('crossover', '2231')]
Resulting state: 2231
Total costs: 0
Value: 4
Correct solution?: False
************************************************
************************************************
uniform_cost
Resulting path:
[(None, '2323'), (('Move Queen 2 to row 4', 2, 4), '2423'), (('Move Queen 3 to row 1', 3, 1), '2413')]
Resulting state: 2413
Total costs: 2
Value: 6
Correct solution?: True
```

```
breadth_first
Resulting path:
[(None, '2323'), (('Move Queen 2 to row 4', 2, 4), '2423'), (('Move Queen 3 to row 1', 3, 1), '2413')]
Resulting state: 2413
Total costs: 2
Value: 6
Correct solution?: True
************************************************
************************************************
astar
Resulting path:
[(None, '2323'), (('Move Queen 2 to row 4', 2, 4), '2423'), (('Move Queen 3 to row 1', 3, 1), '2413')]
Resulting state: 2413
Total costs: 2
Value: 6
Correct solution?: True
************************************************
************************************************
hill_climbing
Resulting path:
[(('Move Queen 4 to row 3', 4, 3), '2413')]
Resulting state: 2413
Total costs: 3
Value: 6
Correct solution?: True
************************************************
************************************************
```

4 Discussion

Completeness: Completeness means does it always find a solution if one exists. For this assignment all of the algorithms find a solution that is true or not. So, we can say all of the algorithms are complete.

Optimality: Optimality means does it always find a least- cost solution. For our project, iterative limited depth first search, greedy, hill climbing random restarts, uniform cost, breadth first, A* are optimal solutions. But limited depth first search, genetic search and hill climbing search algorithms are not the optimal solutions.

Time Complexity: Time complexity is measured with counting number of nodes generated.

Time Complexities of algorithms

Greedy Algorithm

For greedy algorithm, Since there are N queens to be placed, the overall time complexity is O(N^2). For a 7x7 board, Where N =4, the time complexity of the Greedy algorithm for the NQueens problem is (4^2) = O(16).

Iterative limited depth first search

The worst-case time complexity of ILDFS is often exponential, O(b^d), where "b" is the branching factor and "d" is the depth limit. For our specific example with N = 4 and the

initial state '2323', the depth limit "d" will be at most 4, as you need to place 4 queens on the 4x4 board. So, in the worst case, the time complexity could be on the order of O(N^d), which for N=4 and d=4 would be O(4^4) = O(256).

Space Complexity: Space complexity is maximum number of nodes in memory.

Limited Depth First Space Complexity

The space complexity of the "limited depth first" search algorithm for the NQueens problem is influenced by the maximum depth of the search tree. The "limited depth first" search has a space complexity of O(b * d), where b is the branching factor (number of possible actions at each state) and d is the depth limit. In the case of the NQueens problem with a depth limit of 4 (N=4 and the initial state is 2323), the maximum depth of the search tree would be 4. The branching factor at each level is N, as you can place a queen in any of the N rows for each column. Therefore, the space complexity for this specific case would be O(N^d) = O(4^4) = O(256).