BLG435E – ARTIFICIAL INTELLIGENCE – FALL 2018-2019
ASSIGNMENT #1 REPORT

Onat Şahin – 150150129

Q1

(c) Public transportation recommendation system

- Performance measure: The amount of time the recommended routes take
- Environment: Public transportation vehicles and their routes + traffic
- Actuators: An app that recommends routes and vehicles to users
- Sensors: Sensors of the public transportation vehicles that provide data to the backend of the app.

* The environment is PARTIALLY OBSERVABLE. This is because even though we know every vehicle's location, route and working hours, we don't know about possible events in the traffic in advance.
* The environment is STOCHASTIC because of the unpredictability of the traffic
* The environment is EPISODIC because making a recommendation does not have an effect on later recommendations
* The environment is DYNAMIC because the vehicles are always moving.
* The environment is CONTINUOUS because the vehicles are always moving.
* The environment is MULTI-AGENT because there are lots of vehicles and lots of users requiring a recommendation.

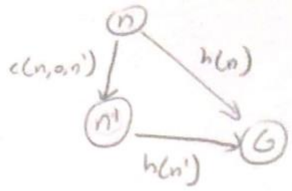
AGENT TYPE: For this environment, the appropriate type of the agent architecture is a UTILITY-BASED AGENT. Since we do not have a specific goal, we need to evaluate the results of the actions using a utility. Utility based agents also keep track of the state and how the world evolves, which are also needed.

Q2

PROOF If a heuristic is consistent, it must be admissible

A heuristic is consistent if

$$h(n) \leq c(n,a,n') + h(n')$$



$k(n)$ → cost of the cheapest path from n to goal node.

A heuristic is admissible if $h(n) \leq k(n)$ for every n.

Induction base case:

If n is a goal node,

$$h(n) = 0 \leq k(n)$$

Induction step:

If n is i steps away from the goal, there exists a node n' that can be created by an action from n. n' is $i-1$ steps away from the goal.

$$h(n) \leq c(n,a,n') + h(n') \quad \text{(consistency rule)}$$

Since the cheapest path from n' to the goal has a cost of $k(n')$:

$$h(n') \leq k(n')$$

$$h(n) \leq c(n,a,n') + k(n') = k(n)$$

↓

Therefore, $h(n) \leq k(n)$ for every node.

If a heuristic is consistent, it must be admissible.

Q3

NOTES:

- Source code files are included in the archive (main.cpp, PegSoltaire.h). The codes should be compiled with C++ 11 (use "g++ -std=c++11 main.cpp" command).
- In the visualizations of the board states, dots represent pegs and circles represent empty holes.

(a) Problem Formulization in a Well-Defined Form

STATES: Different states of the board with different numbers of pegs in different locations.

ACTIONS: Making a valid move with a peg (Pegs can jump orthogonally over an adjacent peg into an empty hole and then the jumped peg is removed.).

INITIAL STATE: The board at the beginning of the game. There is a peg in ever hole except the one in the middle.

GOAL STATE: States of the board where no valid move is possible.

STEP COST: Making a valid move has a unit cost (1). Cost increases as more moves are made.

OPTIMAL SOLUTION: A state where no valid move is possible with maximum number of pegs.

ABSTRACTION: States are represented with 7x7 matrices. -1 is used to represent holes that are out of the board, 0 is used to represent empty holes and 1 is used to represent pegs.

(b) BFS and DFS

Results for these algorithms:

```
PERFORMING DFS...
Found Goal State:

    . . .
    o o o
o . o o o o .
o o o o o o o
. o o o o o o
    o o .
    o o o


The running time: 0.000842 seconds
The number of nodes generated: 134
The number of nodes expanded: 25
The maximum number of nodes kept in the memory: 109
The number of pegs in the found final state: 7


PERFORMING BFS...
Found Goal State:

    . . .
    . o .
. . . . . . .
. o . o . o o
. . . . . . .
    . o .
    . o .


The running time: 0.293349 seconds
The number of nodes generated: 70998
The number of nodes expanded: 8112
The maximum number of nodes kept in the memory: 62886
The number of pegs in the found final state: 26
```

Both of these algorithms are able to find a solution to the problem and reach a goal state. However, it can be seen that while there are 7 pegs in the final state found by DFS, there are 26 pegs in the one found by BFS. This is because DFS is not an optimal search algorithm like BFS. It just keeps making the first possible moves until there are no moves left that can be done. This results in a high cost solution. In contrast, BFS is able to find the optimal solution because it goes over every possible state until it finds the solution with the least amount of moves.

Since DFS keeps making more moves without checking other possible actions, it finds a solution more quickly. This is because making moves certainly leads to a solution (most likely a non-optimal one) in this problem. Because of this quickness, the running time shorter and the number of nodes generated and number of nodes expanded is smaller in DFS. Also, since DFS' space complexity is smaller than BFS (O(bm) compared to O(b^d)), maximum number of nodes kept in the memory is smaller as well.

(c) A* Search

Two heuristic functions I used with the A* search algorithm:

Heuristic-1: Number of pegs that can make a valid move in the given state.
Heuristic-2: Number of possible valid moves in the given state.

```
PERFORMING A* SEARCH WITH HEURISTIC-1 (NUMBER OF PEGS THAT CAN MAKE A MOVE)...
Found Goal State:

    . . .
    . o .
. . . . . . .
. o . o . o o
. . . . . . .
    . o .
    . o .


The running time: 0.001136 seconds
The number of nodes generated: 226
The number of nodes expanded: 50
The maximum number of nodes kept in the memory: 176
The number of pegs in the found final state: 26


PERFORMING A* SEARCH WITH HEURISTIC-2 (NUMBER OF POSSIBLE ACTIONS)...
Found Goal State:

    . . .
    . o .
. . . . . . .
. o . o . o o
. . . . . . .
    . o .
    . o .


The running time: 0.00163 seconds
The number of nodes generated: 205
The number of nodes expanded: 47
The maximum number of nodes kept in the memory: 158
The number of pegs in the found final state: 26
```

It can be seen that A* search algorithm can reach the same goal state that BFS reaches, which is a state where no valid moves are possible and has 26 pegs. This is the case for both heuristics. The reason for this is that A* Search is an optimal search algorithm. However, A* uses a heuristic function to find a solution unlike BFS, which goes through every state in the search tree level by level until it finds a goal state. Because of this, A* search algorithm uses less resources than BFS. Its runtimes are shorter, number of nodes generated and expanded are smaller, and it uses way less memory.

Comparing the two heuristic functions, we can see that heuristic-2 performs slightly better than heuristic-1 in terms of node counts and memory usage. This is because heuristic-2 gives a higher cost than heuristic-1 all the time and dominates it. Therefore, heuristic-2 is better than heuristic-1 for searching. However, the difference is very small. I think the reason for this is that both heuristics are logically tied to each other and they never conflict. Since difference is small, their runtimes are approximately the same as well.

(d)

If the objective is changed to finding a state where only one peg remains, that would make the running times of BFS, DFS and A* search algorithms longer. This is because the goal states will be in the lowest level of the search tree and number of goal states will be very small. For this reason, the algorithms will have to search deeper in the search tree. This means that there will be a lot more nodes discovered and expanded. Because of this, memory usage and runtime will increase.