ARTIFICIAL INTELLIGENCE HOMEWORK 1 REPORT

MEHMET BARIŞ YAMAN
150130136

Q1) PEAS of the public transportation and recommendation system:

Performance Measures

→ Cost of the transports

→ Time that users spent on transportation

→ Confidence of the users in selected transports

Environment

→ Mobile Application

→ Internet's used tourism web sites

Actuators

→ Key entry

→ Cursor

→ Finger

Sensors

→ Website interface

→ Application interface

→ Browser

Characteristics

- Partially observable
- Multiagent
- Stochastic
- Sequential
- Dynamic
- Continuous

Appropriate Type of the Agent

Goal-Based Agent since the agent always tries to find the goal, which is the optimal transportation way depending on the user input.

Q2) Proof of the theorem "If a heuristic is consistent, it must be admissible": (Proof by induction) – Select t as the goal node

$h(t) = 0$ since the optimal cost from one node to itself is 0

Base Case: Assume that node t is the child of node n. If the heuristic is consistent, then $h(n) \leq h(t) + c(n,t) = 0 + c(n,t) = c(n,t)$. Since $h(n) \leq c(n,t)$ then, h is admissible

Induction: Assume n' is in the set of successor nodes of n. Using the consistency then $h(n) \leq h(n') + c(n,n')$. Consider $h^*(n')$ is the optimal cost of the reachability of the node n'. Then, $h(n') \leq h^*(n')$. Therefore $h(n) \leq h(n') + c(n,n') \leq h^*(n') + c(n,n') = h^*(n)$. This implies that $h(n) \leq h^*(n)$ and h is admissible.

– Construction of an admissible heuristic, but not consistent:

We can consider 8-puzzle problem again and construct the heuristic by using the Manhattan distance of the 1,2,3,4,5,6,7,8 to their goal-state locations.

But this time, instead of considering the Manhattan distance of all the nodes, we can only consider a set of them such as 2,5,7. Therefore the heuristic considers the Manhattan distance of only 2,5,7 whose cost is less than the optimal solution including the Manhattan distance of all nodes, the constructed heuristic is thus still admissible.

But since the constructed heuristic only cares 2,5,7, we can not accept that there is a relationship between the heuristic estimates of each node and therefore the constructed heuristic is not consistent.

Q3) a) Tree Search Version of BFS and DFS

Since we are implementing tree search version of BFS and DFS algorithms, we could not make the agent reach the goal state. Because different than the graph search version of DFS and BFS, we do not make the agent control whether the state is reached before or not. When the agent does not check the state is visited or not, it passes same states again and again and therefore an infinite loop is occurred. Because of the infinite loop we can not return the directions in BFS and DFS. Therefore those algorithms could not find the solution since they are in tree versioned.

b) A* Search with null Heuristic and Manhattan Distance Heuristic

In this part we are asked to implement A* search in two versions. The first version makes the heuristic function return always 1 and the second one makes it return the Manhattan distance. In this part I used Simple AI Library, which can be found on link:

https://github.com/simpleai-team/simpleai

According to my implementation if we want to run the A* function with heuristic 1 that command is used:

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar

For the A* with Manhattan Distance heuristic, this command is used:

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

After running and comparing these two implementations, I got these results:

Comparison	**Null Heuristic**	**Manhattan Distance Heuristic**
# of nodes created	1245	1105
# of nodes expanded	621	549
Max # of nodes kept in memory	1866	1654
Running time	0.1 seconds	0.1 seconds

The A* Algorithm works in Simple AI Library as:

set the memory and initial node
initial node is added to BoundedPriorityQueue
while BoundedPriorityQueue is not empty:
    First item is selected
    If the first item is goal state
        print and return the node
    else
        print the node
    add nodes state to memory
    get successors of the node
    for each successor
        use selected heuristic (1 or calculate manhattanDistance
        if the state of the successor is not in memory and the return of the heuristic is 0
            add the node to the BoundedPriorityQueue