BLG435E - Assignment #1

Ramazan Yetişmiş-150190708

November 10, 2021

Question 1

Performance: Time, safety of the deliveries,optimal path,energy consumption,battery life
Environment: Road,other robots, pedestrians,road signs,obstacles,pavements,workers
Actuators: Steering,accelerator,brake,horn,lever,bags,wheels
Sensors: Camera ,GPS,Accelero-meter,Speedometer, motion sensor, engine sensor,distance sensor,keyboard
Task Environments:

Observable	**Deterministic**	**Episodic**	**Static**	**Discrete**	**Agents**
Partially	Stochastic	Sequential	Dynamic	Continuous	Multi

Agent Type:
This is a utility-based agent because it does not only delivers the products, it delivers the the products in a safe and fast way. For instance if the time is important it tries to find the shortest path, if the safety the of the product is important then it will consider the safest path. So by just looking at the utility based agents description we can conclude that this type of agents can be named as **utility-based agent**

Question 2

Consistency $\Longrightarrow h(n) \leq c(n, n+1) + h(n+1)$

Base Case: $n =$ **Goal state**

$$h(n-1) \leq c(n-1, n) + h(n)$$

We know that n is a goal state then $h(n) = h^*(n)$.Thus the above equation is now

$$h(n-1) \leq c(n-1, n) + h^*(n)$$

and given that $c(n-1,n) + h^*(n) = h^*(n-1)$, that can be concluded from this equation that this is the definition of the admissibility:

$$h(n-1) \leq h^*(n-1)$$

Inductive Step: Try for the $n-2$ case.
The cost can be written as

$$h(n-2) \leq c(n-2,n-1) + h(n-1)$$

By looking to the base case we know that:

$$h(n-2) \leq c(n-2,n-1) + h(n-1) \leq c(n-2,n-1) + h^*(n-1)$$

$$h(n-2) \leq c(n-2,n-1) + h^*(n-1)$$

So again we can estimate $c(n-2,n-1) + h^*(n-1) = h^*(n-2)$ so we can see that:

$$h(n-2) \leq h^*(n-2)$$

So using proof by inductive method we can conclude that this will hold for all nodes **therefore the consistency implies admissibility.**

Example:

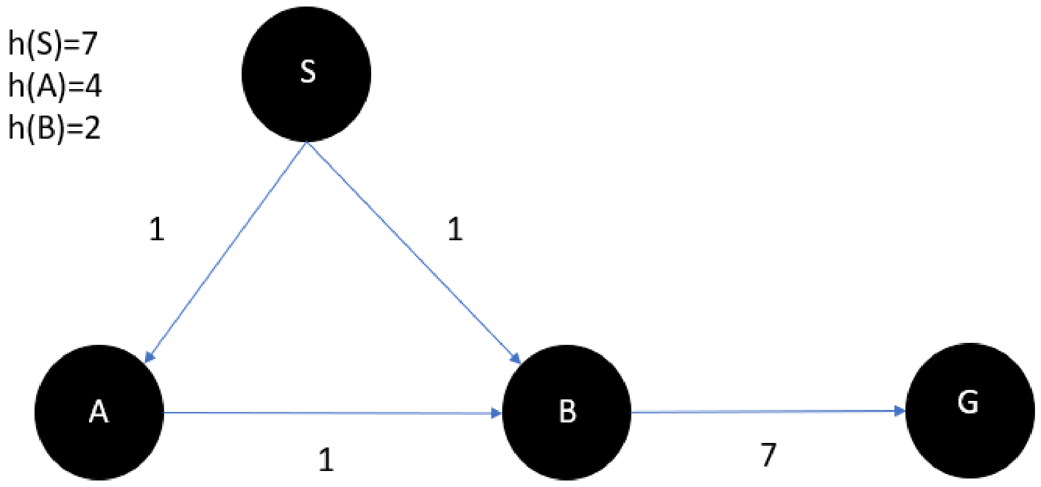

Figure 1: $y_1[n]$

we are going t check the consistency dependency $h(S) \leq c(S,a,B) + h(B)$ so we can see that $h(S) = 7 \nleq c(S,a,B) = 1 + h(B) = 2$ so we can say that this **h(n) is admissible but not consistent.**

Question 3

BFS

In this part I will briefly mention about the BFS algorithm and 2 different implementation method.

BFS-Tree-Version

Listing 1: Tree version BFS

```
open=new Queue()
root(x=player_x ,y=player_y ,Parent=None)
open.push(root)
while open is not empty:
    current=open.pop()
    if current==goal :
        return sequence
    for successor in nearbyPoint:
        if succesor not wall:
        open.push(successorNode)
```

This is the basic pseudo-code of the tree-search version of our Bomber-man.So here we can make some comments,as we can see our data structure for the frontier is the Queue so this search is being done **level by level**. That means first we are exploring the first levels and look if that is the goal state, if not we are adding the appropriate successors nodes to the frontier.
Run Time:

Listing 2: Output for Level 1

```
['R', 'R', 'D', 'D', 'D', 'D', 'R', 'R', 'D', 'D', 'R']
                        BFSAgent decided sequence length:11
                        --- BFS Agent Statistics ---
                        elapsed time step:11
                        number of generated nodes:2486
                        number of expanded nodes:1248
                        maximum number of nodes kept in memory:1240
                        elapsed solve time:0.1410679817199707
                        WON
```

1)Tree Implementatıon

In level 2 the function does not converge to the solution in a limited time. The run time is (b^d).In tree search method we know repetitions are allowed so the memory and complexity increases, the algorithm can not find the path with limited sources. Because level 1 is not big enough the BFS can converge easliy.

2)Graph Implementatıon

Listing 3: Graph version BFS

```
open=new Queue()
closed=[]
root(x=player_x,y=player_y,Parent=None)
open.push(root)
while open is not empty:
    current=open.pop()
    if current==goal :
        return sequence
    for successor in nearbyPoint:
        if succesor not wall and it is not explored before:
        open.push(successorNode)
    closed.append(current)
```

I have to point out that in my code closed list keeps the level matrix in order to check if it is explored or not, because this is easier than copying and controlling the node, because node is more complex and bigger(with respect to size).Now there is only one difference from the tree search version it is the new list called closed that keeps track of the states, basically it helps us to determine if we visited this state previously if we visited it then we are not going to consider it again.

Listing 4: Outputs

```
          LEVEL−1
−−−− BFS Agent Statistics −−−−
elapsed time step:11
number of generated nodes:15
number of expanded nodes:14
maximum number of nodes kept in memory:2
elapsed solve time:0.0023288726806640625
WON
         LEVEL−2
−−−− BFS Agent Statistics −−−−
elapsed time step:211
number of generated nodes:623
number of expanded nodes:626
maximum number of nodes kept in memory:8
elapsed solve time:2.0237066745758057
WON
```

Here we can make some conclusions such that, with the graph version even we are in level 2 we can reach the optimal in a limited time. We prevent the unnecessary repetitions so the algorithm became more efficient. The other conclusion is in level-1 outputs we can see that created nodes are less in graph version and as well as memory usage.

DFS

We know that DFS is not a complete algorithm it finds the goal path when the number of nodes are limited and repetitions are not allowed.So we will examine this conditions in the following sections

1) Tree Version

We know that DFS is not a complete algorithm and when I used the tree method it diverges. So I can not get the output path in both levels. As a result as we know, although level one is very small DFS could not manage to find the goal, because it stuck with the unnecessary loops. I will not provide any pseudo-code for this part because the only difference with the BFS algorithm is the choice of the data structure in this part we will use **Stack data structure**.

2)Graph Version

From the algorithm name that can be deduct that the algorithm will search the deepest nodes first then it will came back to the upper levels. So when repetition allowed it stuck at the bottom and diverges.**Stack** helps us to look for the last added notes (depth). When I used the graph method the algorithm finds the goal state.

Listing 5: Outputs

```
                              LEVEL-1
['R', 'R', 'D', 'D', 'D', 'D', 'R', 'R', 'D', 'D', 'R']
                    DFSAgent decided sequence length:11
                    ---- DFS Agent Statistics ----
                    elapsed time step:11
                    number of generated nodes:13
                    number of expanded nodes:14
                    maximum number of nodes kept in memory:3
                    elapsed solve time:0.002101898193359375
                    WON

                             LEVEL-2
                   ---- DFS Agent Statistics ----
                    elapsed time step:211
                    number of generated nodes:426
                    number of expanded nodes:389
                    maximum number of nodes kept in memory:40
                    elapsed solve time:1.1621520519256592
                    WON
```

Comparison BFS &DFS

I already compared them but I will summarise the messy parts now. BFS algorithm is a **complete & optimal** algorithm and it search level by by and it find the goal state but it consumes a lot of time and memory, however DFS is **not a/n (complete & optimal)** algorithm we see from the tree version level 1 example. But if there is a limited node and repetitions are not allowed then DFS finds the goal state too,but this may not be the best path. So when we looked at the of **expanded nodes and generated nodes** DFS algorithm is better than BFS because $SP_{BFS}(b^d) > SP_{DFS}(b \times m)$ sp denotes for space complexity.

Important NOTE:
If you want to see the tree version of the algorithms you have to comment out the parts in the source file.

A*

Listing 6: Outputs

```
open=new heap()
root(x=player_x ,y=player_y ,Parent=None, FinalCost=givenCost+HeuristicCost)
open.push(root)
while open is not empty:
    current=open.pop()
    if current==goal :
        return sequence
    for successor in nearbyPoint:
        successor=Node(Parent=current , FinalCost=givenCost+HeuristicCost)

        if a node for this nearbyPoint has been created before , then
            - if successor better than oldNode, then
                    closed.remove(successor)
            - else
                - skip to the next nearbyPoint
        if succesor not wall:
            open.push(successorNode)
        closed.push(current)
```

This algorithm is different than the previous algorithms because it uses and additional cost called heuristic. We already explained what is admissible and consistent heuristic. Again the general structure is similar to the previous search algorithms, however we have some additional cost that comes from h(n)and g(n)(actual given cost) **f(n) = g(n) + h(n)**. A* algorithm is complete and optimal it guarantees the shortest path. In this search method h(n) can determines the efficiency such if $h_2 > h_1$ then h_2 dominates h_1 , and is better for search.

H(n)=Manhattan Distance:

Listing 7: Outputs

```
                              LEVEL-1
['R', 'R', 'D', 'D', 'D', 'D', 'R', 'R', 'D', 'D', 'R']
                    ----- A* Agent Statistics -----
                         elapsed time step:11
                         number of generated nodes:14
                         number of expanded nodes:12
                         maximum number of nodes kept in memory:4
                         elapsed solve time:0.0017237663269042969
                         WON


                              LEVEL-2
                    ----- A* Agent Statistics -----
                         elapsed time step:211
                         number of generated nodes:557
                         number of expanded nodes:550
                         maximum number of nodes kept in memory:12
                         elapsed solve time:1.6230688095092773
                         WON
```

H(n)=Euclidean Distance:

Listing 8: Outputs

```
                              LEVEL-1
['R', 'R', 'D', 'D', 'D', 'D', 'R', 'R', 'D', 'D', 'R']
                    ----- A* Agent Statistics -----
                     elapsed time step:11
                     number of generated nodes:14
                     number of expanded nodes:13
                     maximum number of nodes kept in memory:4
                     elapsed solve time:0.0019047260284423828
                     WON



                              LEVEL-2
                    ----- A* Agent Statistics -----
                     elapsed time step:211
                     number of generated nodes:568
                     number of expanded nodes:560
                     maximum number of nodes kept in memory:10
                     elapsed solve time:1.70865797996521
                     WON
```

Evaluation:
From the outputs I can conclude that if we use Euclidean distance our memory consumption is definitely lower. However it **expands and generates** more node during the process, because of this things the run time becomes more

Overall Evaluation:
1) Generated & Expanded Node:
$DFS < A * h_1 < A * h_2 < BFS$
here for the BFS and DFS graph version are counted.
2) Frontier size (number of node kept in the memory):
$BFS < A * h_2 < A * h_1 < DFS$
3) Run time:
$DFS < h_1 < h_2 < BFS$

References For Part 3
1-Pseudo Code-Game Design & Development, Full Sail Real World Education "www.fullsail.com"
2-https://learning.edx.org/course/course-v1:HarvardX+CS50AI+1T2020/home
3-https://github.com/ecemkonu/AI-Search-Algorithms/blob/master/agent.py