Introduction To Artificial Intelligence

HW 1

Maroon Ayoub – 315085597

Yara Kassem - 212108880

Question 1:

1.1: The search problem for the taxi environment is $\{S, O, I, G\}$:

S: The state space consists of the 500 states in the environment, while a state is defined by the location of the taxi, the location of the passenger, and the passenger's destination.

A state is a 4-tuple $< taxi\_row, taxi\_column, passenger\_location , destination >$:

$taxi\_row \in \{0,1, \ldots ,24\}$: *the row that the agent is at.*

$taxi\_column \in \{0,1, \ldots ,24\}$: *the column that the agent is at.*

$passenger\_location \in \{R , G, B, Y, in\_taxi \}$: *the location of the passenger.*

the location can be one of the positions R, G, B, Y or inside the taxi.

$destination \in \{R, G, B, Y\}$: *the destination of the passenger.*

In each state the taxi is at one of the 25 locations on the map , the passenger can be at one of the R,G,B,Y locations or in the taxi, and the passenger's desired destination can be one of the R,G,B,Y locations , therefore there are $25 * 5 * 4 = 500$ states in the state space.

O: The set of operators is $= \{o_0, o_1, o_2, o_3, o_4, o_5\} \ s.t. \ o_i: S \rightarrow S \cup \{\emptyset\}$.

$o_i \ (0 \leq i \leq 3)$ Moves the taxi from the current position to the correct direction if there is no wall (pipe) between them with a penalty of -1 points. o_0 Moves the taxi to the south, o_1 moves the taxi to the north, o_2 moves the taxi to the east and o_3 moves the taxi to the west.

o_4 Is the pickup operation. It picks up the passenger with a penalty of -1 points if there is a passenger at the current location, or with a penalty of -10 points if there is no passenger at the current location.

o_5 Is the drop-off operation. If it is performed with no passenger aboard, the environment penalizes -10 points. Otherwise, if the passenger is dropped at their destination, the environment rewards 20 points, if the passenger is not dropped at their destination it penalizes -1 points.

I: At the start state the taxi can be at any of the 25 locations on the map, the passenger and their destination can be at any of the R, G, B, Y locations.

G: The goal state is when the passenger is dropped at the destination.

1.2 : $\mathrm{Domain}(\mathrm{operator} - \mathrm{north}) = \mathrm{Domain}(\mathrm{o}_1) = \{s \in S \mid o_1(s) \neq \emptyset\}$, are all of the states that the agent is not at the top row of the map in them, and that don't have a wall (pipe) between the position of the agent and the position north of it (directly on top of it in the map).

Therefore $\mathrm{Domain}(\mathrm{o}_1) =$

$\{S \setminus \{< 0, taxi\_column, passenger\_location, destination >\}\}$ In the given environment.

Commented [YK1]: Should we return the same state if an illegal action was performed?
Ask.

1.3: $Succ(328) \overset{\text{def}}{=} \{s' \in S | \exists o \in O \; s.t. \; [328 \in Domain(o) \land o(328) = s']\} = \{428{,}228{,}348\}$

1.4: infinite – solution may not be reached in the worst case if it gets stuck in loops.

1.5: the agent can reach the solution in 10 actions in the best case, since the passenger is at Y and the destination is R , the shortest way to reach the solution is this list of actions: $north \rightarrow west \rightarrow south \rightarrow south \rightarrow pickup \rightarrow north \rightarrow north \rightarrow north \rightarrow north \rightarrow dropoff$.

1.6: since the agent has to perform this list of actions: $north \rightarrow west \rightarrow south \rightarrow south \rightarrow pickup \rightarrow north \rightarrow north \rightarrow north \rightarrow north \rightarrow dropoff$, the list of operators is: $o_1 \rightarrow o_3 \rightarrow o_0 \rightarrow o_0 \rightarrow o_4 \rightarrow o_1 \rightarrow o_1 \rightarrow o_1 \rightarrow o_1 \rightarrow o_5$, and the total reward is $(-1) * 9 + 20 = 11$.

1.7: yes, it is possible, for instance $328 \rightarrow 348 \rightarrow 448 \rightarrow 428 \rightarrow 328$.

<u>Question 2:</u>

2.2: running BFS-G from state 328 leads to visiting (closing) 31 vertices.

2.3: before opening a vertex we check whether it has been already opened or closed, if it has been then we do not open it again. That means a vertex can be closed only once, therefore all the 31 vertices that were closed after running BFS-G from state 328 are distinct.

2.4: BFS's advantage over DFS in this environment: BFS is optimal. It is guaranteed that it returns the least cost path (explanation in 2.5), while DFS is not optimal.

DFS's advantage over BFS in this environment: memory footprint.

2.5: yes, BFS is optimal in this taxi problem. Proof:

All the actions cost -1 points except for an illegal pickup or drop off which cost -10 points, and for the final drop-off at the destination which reward 20 points. Since the drop-off at the destination is the last action to be performed, the goal state is discovered before the action is performed (when the neighbors of the previous states are discovered) the reward is not relevant. Moreover, an illegal pickup or drop-off does not introduce a new state, it leads to a leaf which is already in the closed list, resulting in stopping the search in this branch, therefore these actions are also not relevant.

All the relevant actions cost the same (-1 points), therefore we can run BFS as we learned, and the shortest path that is returned is guaranteed to be the least cost path.

Question 3:

3.2: running DFS from state 328 leads to closing 100 vertices.

3.3: before opening a vertex, we check whether it has been already closed. If it has been then we do not open it again.
That means a vertex can be closed only once, therefore all the 100 vertices that were closed after running DFS from state 328 are different.

<u>Question 4:</u>

4.2: running ID-DFS from state 328 leads to closing 393 vertices when we check if a node hits the depth-limit before if it is a goal node. If the checks were reversed, the number would be 308.

4.3: running ID-DFS from state 328 leads to closing 35 vertices (or 31, respective to the above).

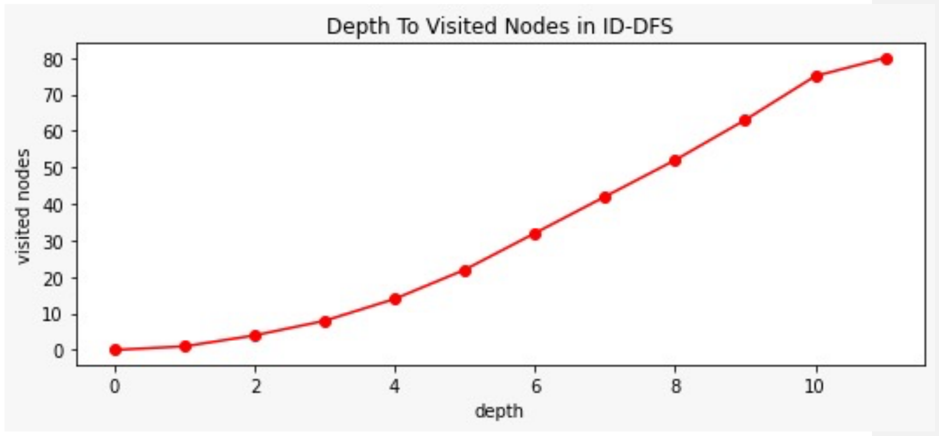
4.4: ID-DFS's advantage over DFS: if a solution was found using ID-DFS then it is optimal (same reason as BFS), while if it was found using DFS it is not guaranteed.

ID-DFS's disadvantage over DFS: the number of visited vertices in ID-DFS is much larger than the number of visited vertices in DFS (higher runtime).

4.5: ID-DFS's advantage over BFS: memory footprint.

ID-DFS's disadvantage over BFS: the number of visited vertices in ID-DFS is larger than the number of visited vertices in BFS (higher runtime).

4.6: the deeper the search in ID-DFS the more vertices are visited.



Question 5:

5.3: yes, it is possible, if a better path to the state was found then it is opened again.

We can see this at this line in the algorithm shown in lecture:

“OPEN.insert(n_curr) “ // n_curr <- node in CLOSED with state s

----

Redefining the cost by using the f function:

$$f\ (reward) = \begin{cases} |reward| & ,reward \neq 20 \\ 0 & ,reward = 20 \end{cases}$$

The true cost - $h^*(s)$ – is the minimal sum of f (reward) of the rewards of all actions performed to reach the goal.

5.5: 1) greedyHeurisitc: Admissible.

Proof: for every $s \in S\ s.t\ s \neq goal\ , h^*(s) \geq 1 = h(s) \geq 0$.for $s \in S\ s.t\ = goal\ , h^*(s) = 0 = h(s)$.

2) ManhatanSumHeurisitc: Admissible.

Proof: first we acknowledge that in this specific environment Manhattan distance is always less than or equal to the true cost. Dividing to two cases:

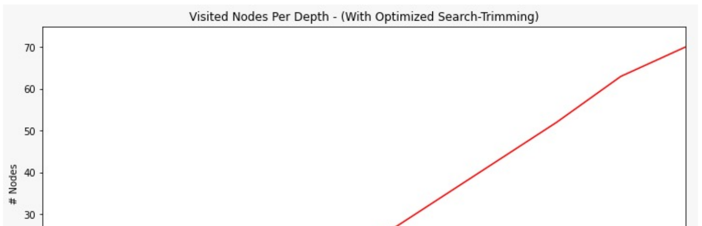
1. If the passenger has been picked up : $0 \leq ManhatanSumHeurisitc(s) = MD(s, loc_{passenger}) + MD(loc_{passenger}, loc_{drop-off}) =_a 0 + MD(loc_{passenger}, loc_{drop-off}) \leq_b true\ cost\ to\ reach\ destination\ from\ current\ location = h^*(s)$

(a: the location of the taxi is the same location of the passenger since the passenger has already been picked up)

(b: Manhattan distance is less or equal to true cost)

2. If the passenger has not been picked up yet: $0 \leq ManhatanSumHeurisitc(s) = MD(s, loc_{passenger}) + MD(loc_{passenger}, loc_{drop-off}) = true\ cost\ to\ reach\ pick-up\ location\ from\ current\ location + true\ cost\ to\ reach\ destination\ from\ pick-up\ location = h^*(s)$

(Manhattan distance is less or equal to true cost, moreover the taxi has to –at least- reach pick up location then reach the destination from the pickup location)



We proved that the heuristic is admissible overall.

3) PickupSumHeuristic(s): Inadmissible.

Counter example: if $s = goal\ state\ , pick-up\ location\ is\ R\ and\ drop-off\ location\ is\ Y\ then\ h^*(s) = 0$,

$$but\ \text{PickupSumHeuristic}(s) = \frac{MD(s, loc_{pick-up}) + MD(loc_{pick-up}, loc_{drop-off})}{25} > 0$$

4) PickupMultHeuristic: Inadmissible.

Counter example: if $s = 79\ , pick-up\ location\ is\ Y\ and\ drop-off\ location\ is\ G\ then\ h^*(s) = 1$,

$$but\ \text{PickupMultHeuristic}(s) = \frac{MD(s, loc_{pick-up}) * MD(loc_{pick-up}, loc_{drop-off})}{25} = \frac{7 * 8}{25} = 2.24 > 1$$

The most informative heuristic between the 4 given heuristics is the second one- ManhatanSumHeurisitc.

for every $s \in S$ $\text{ManhatanSumHeurisitc}(s) \geq \text{greedyHeurisitc(s)}$ $(similar\ to\ what\ we\ learned\ in\ class)$ therefore, it is more informative.

PickupMultHeuristic and PickupSumHeuristic are inadmissible heuristics, so it is trivial that ManhatanSumHeurisitc is more informative.

5.7: 14 vertices were closed when A^* was performed using ManhatanSumHeurisitc, meanwhile the number of vertices that were closed when BFS was performed is 31, when DFS was performed is 100 and when ID-DFS was performed is 393 (35 different vertices).

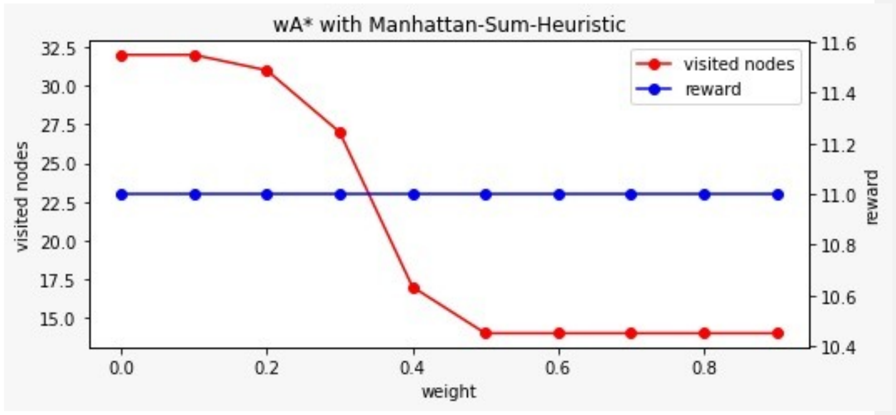
$$\frac{number\ of\ closed\ vertices\ in\ A^*}{number\ of\ closed\ vertices\ in\ BFS} = 0.45$$

$$\frac{number\ of\ closed\ vertices\ in\ A^*}{number\ of\ closed\ vertices\ in\ DFS} = 0.14$$

$$\frac{number\ of\ closed\ vertices\ in\ A^*}{number\ of\ closed\ vertices\ in\ ID\ DFS} = 0.0356$$

$$\frac{number\ of\ closed\ vertices\ in\ A^*}{number\ of\ different\ closed\ vertices\ in\ ID\ DFS} = 0.4$$

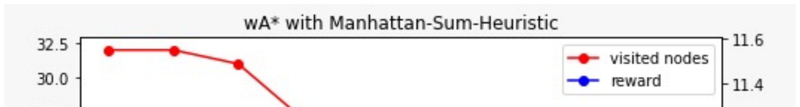
5.9:



5.10: the graph describes that changing the weight does not affect the reward, it is fixed at 11, but increasing the weight decreases the number of closed nodes when running A^*.

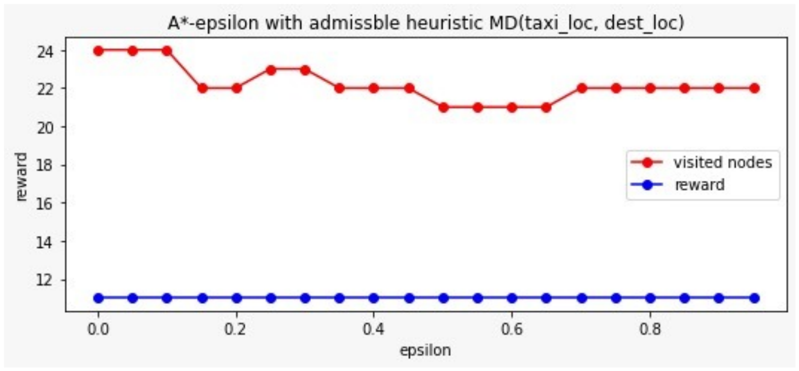
Therefore, it is better to run A^* with a weight larger the 0.5, since decreasing the number of closed nodes decreases the runtime, and there is no downside to increasing the weight since the reward is not affected by it.

We can see in this example that the principle learned in class is not necessarily always true, the reward with weight=0.6 is not better than the reward with weight=0.8 (they are equal), and the number of closed nodes with weight=0.8 is not less the number of closed nodes with weight=0.6 (they are equal).

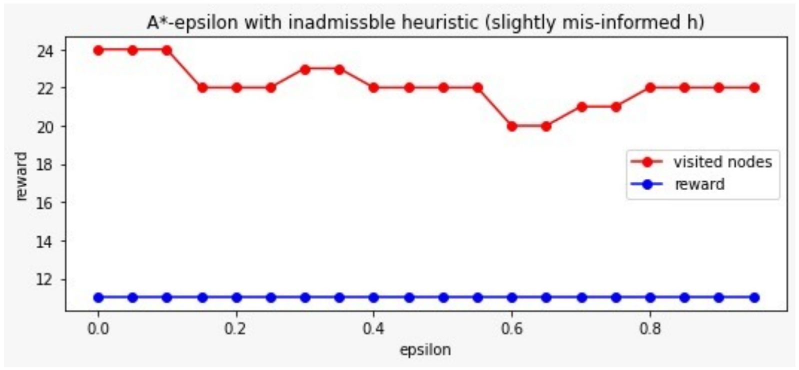


Question 6:

6.3: The admissible heuristic is MD(taxi location, destination), the result we got using it is 11.



The inadmissible heuristic is MD(taxi location , destination with offset 1) with result 11.



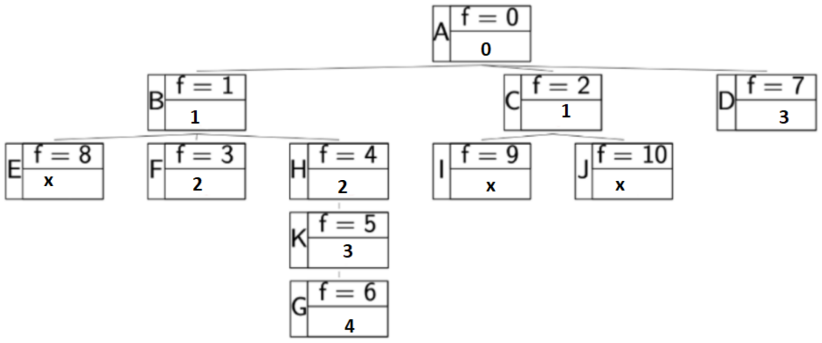
The number of closed vertices did decrease, since A* epsilon when epsilon is 0 is practically A*, it is clear in the graphs.

We expect the quality of the results using $A^{*}epsilon$ to be worse than that in the results using A^{*} and the number of visited vertices using $A^{*}epsilon$ to be less than the number of vertices using A^{*}, due to how we pop a node from the frontier.
In A^{*}, the vertices are ordered in open using f which is a combination of h and g, while when using epsilon, we expand the expansion selection options then choose the vertex with the minimal h value. Meaning we give more weight to the heuristic compared to A^{*}.

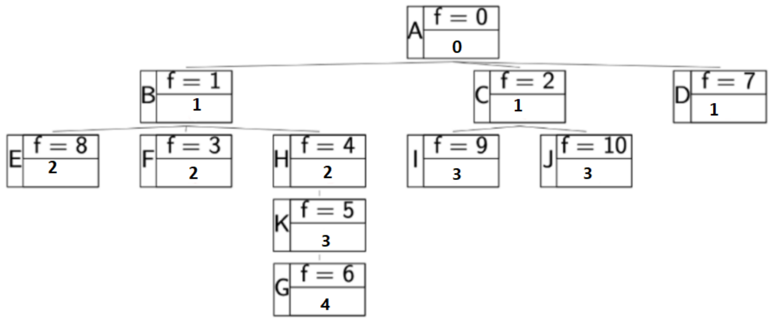
Part D – exam style question

1. Answers
   I. A. Complete: in the worst case, all nodes are opened and tested. If one node is a goal node, then Should-Return allows its finding.
   B. Inadmissible: due to the parallelism, we cannot guarantee that the optimal path would be found in cases where there is more than one path to the best goal node, since the worker which holds a node on the optimal path may get preempted, allowing a worker with a state that is NOT on the optimal path to be expanded first.
   II. A. Incomplete: in a case where there is only one path to the goal node, there is a possibility that other workers are busy when the goal state is popped, and the function Should-Return is executed. Meaning goal node is lost.
   B. Inadmissible: since it is not complete it cannot be admissible.
   III. A. Complete: goal node can either be not opened at all or in OPEN, because it is inserted back to OPEN even if false should be returned, which means it keeps getting evaluated until all workers are idle.
   B. Inadmissible: assume two workers A and B hold two states (A and B respectively) that lead (directly) to a single goal state. Assume that A holds the state on the optimal path, while B does not. It is possible for worker B to finish up first, then for the master to pop the goal node (whose parent is state B), test the node and then get preempted before evaluating the "return" instruction in Should-Return. Finally, worker A would complete its job, and let the master continue the evaluation of "return All-Idle", which would be true.
   IV. B. Admissible: the first line guarantees that the parent of the goal node in the optimal path is either inserted in OPEN or have been evaluated (optimal goal node is inserted to OPEN), the second line guarantees that the optimal goal node stays in OPEN, and the third line guarantees that the node is not returned unless it is the optimal goal node.
   A. Complete: it is admissible therefore it is complete.
2. Answers

I.



II.



Note that there are more options in the first part, depending on the scheduling of the workers. Where E, I and J could have been replaced by:
(4, X, X), (4, 5, X), (4, 5, 6).