Introduction To Artifical Intelligence - HW1

Yoni Ben-Zvi 203668900 Danny Priymak 307003434

Question 1

The requested table is produced by the following code:

```
from tabulate import tabulate
import math
list1 = [] for x in range(1, 11):
        list2 = [x, math.factorial(x), math.factorial(x)*(5**(x-1))]
        list1.append(list2)
print(tabulate(list1, headers=['k','l = 0','l = 5'], tablefmt='orgtbl'))
```

and its output is:

```
|  k |  l = 0 |          l = 5 |
|----+--------+-----------------|
|  1 |      1 |               1 |
|  2 |      2 |              10 |
|  3 |      6 |             150 |
|  4 |     24 |            3000 |
|  5 |    120 |           75000 |
|  6 |    720 |         2250000 |
|  7 |   5040 |        78750000 |
|  8 |  40320 |      3150000000 |
|  9 | 362880 |    141750000000 |
| 10 | 3628800 |  7087500000000 |
```

Question 2

The branching factor's extremum values are:

Maximum $= k + l$ because if our graph is a complete graph, each vertex is connected to all other gas stations and all other delivery points. If the current vertex is v_0, then it is connected to all other $k + l$ vertices of the graph. This is the maximal number of vertices it can be connected to using the operators defined in the exercise.

Minimum $= 1$ because if the graph is 1-regular (all outdegrees are 1) then the maximal outdegree of each vertex is 1, and therefore the graph's branching factor is 1.

Question 3

Yes. Consider a graph that contains two gas station vertices f_1, f_2 that are connected to each other, and the distance between them is less than the maximal distance the scooter can go when its fuel tank is full. Suppose the scooter is currently at f_1. Therefore, its fuel tank is full. Let us denote this state as S_1. Suppose it chooses to go to f_2. Once it arrives at f_2, its fuel level is once again full. Lastly, suppose the scooter chooses to go back to f_1. Once it arrives there (for the second time), its fuel capacity is once again full. Note that the current state is identical to S_1, as The sets T, F have not changed during this path's traversal, and the fuel levels were constant. Therefore, we have found a directed cycle in the graph.

Question 4

Let us first distinguish between the start state $(v_0, d_0, \text{Ord}, \emptyset)$ and all other states.

Since we assume no order or gas station can be placed in v_0, the only time the scooter will be at v_0 is when the algorithm starts. Hence, we can count this state only once.

Once we have the distinction above, the number of options for the scooter's location is $k+l$, as it can only stop at gas stations (which there are l of) or delivery drop-off locations (which there are k of).

As for the fuel level d, if we assume the possible fuel levels are real values between 0 and d_{refuel} in, say, double-precision, then the number of options is $2^{64} = 18446744073709551616$ (with maximal resolution possible with respect to the real value range $[0, d_{\text{refuel}}]$).

As was pointed out in the exercise, we can keep track of only one of the sets T, F, since the other is its complimentary set with respect to $[k] = \{1, \ldots, k\}$. Without loss of generality let us keep track of only T. Hence, the number of possible sets T is $|\mathcal{P}([k])| = 2^k$, where $\mathcal{P}()$ denotes the power set.

To summarize, the resulting number of states is given by the formula

$$((k+l) \cdot (2^{64}) \cdot 2^k) + 1$$

Question 5

Yes. Assume that $|T| \geq 2$ and that the scooter has just enough gas to reach a certain delivery drop-off point t_i from its current location, and has exactly zero fuel left upon arrival at t_i. The operation $|T| \leftarrow |T| - 1$ is computed, and the scooter cannot go further to any delivery drop-off point nor gas station in the graph, and the new state it's at is not a goal state since $|T| \geq 1$.

Question 6

$$\text{Succ}_1\left((v_1, d_1, T_1, F_1)\right) = \left\{ (v_2, d_2, T_2, F_2) \in S : \begin{array}{c} v_2 \in \text{Ord} \\ d_2 = d_1 - \text{Dist}(v_1, v_2) \quad \wedge \quad d_1 - \text{Dist}(v_1, v_2) \geq 0 \\ \exists i \in [k]: \quad i \in T_1 \quad \wedge \quad T_2 = T_1 \setminus \{i\} \quad \wedge \quad F_2 = F_1 \cup \{i\} \\ \text{There exists a directed path } v_1 \to \cdots \to v_2 \text{ on the map} \end{array} \right\}$$

$$\text{Succ}_2\left((v_1, d_1, T_1, F_1)\right) = \left\{ (v_2, d_2, T_2, F_2) \in S : \begin{array}{c} v_2 \in \text{GasStations} \\ d_2 = d_{\text{refuel}} \quad \wedge \quad d_1 - \text{Dist}(v_1, v_2) \geq 0 \\ T_1 = T_2 \quad \wedge \quad F_1 = F_2 \\ \text{There exists a directed path } v_1 \to \cdots \to v_2 \text{ on the map} \end{array} \right\}$$

$$\text{Succ}((v_1, d_1, T_1, F_1)) = \text{Succ}_1\left((v_1, d_1, T_1, F_1)\right) \cup \text{Succ}_2\left((v_1, d_1, T_1, F_1)\right)$$

Question 7

If we ignore the fuel constraint and assume that d_0 is very big such that the scooter does not need to refuel during its trip, the goal state minimal depth must be at least the number of delivery drop-off points k, as the scooter must go through all of them to get to a goal state. If we reconsider the fuel constraint, each refuel operation adds one level of depth to the search, hence increasing the depth. Therefore, the minimal depth is k.

Question 8

load_map_from_csv: 1.52sec

Solve the map problem. Map(src: 54 dst: 549) UniformCost time: 0.59 #dev: 17355 total_cost: 7465.52897 |path|: 137 path: [54, 55, 56, 57, 58, 59, 60, 28893, 14580, 14590, 14591, 14592, 14593, 81892, 25814, 81, 26236, 26234, 1188, 33068, 33069, 33070, 15474, 33071, 5020, 21699, 33072, 33073, 33074, 16203, 9847, 9848, 9849, 9850, 9851, 335, 9852, 82906, 82907, 82908, 82909, 95454, 96539, 72369, 94627, 38553, 72367, 29007, 94632, 96540, 9269, 82890, 29049, 29026, 82682, 71897, 83380, 96541, 82904, 96542, 96543, 96544, 96545, 96546, 96547, 82911, 82928, 24841,

24842, 24843, 5215, 24844, 9274, 24845, 24846, 24847, 24848, 24849, 24850, 24851, 24852, 24853, 24854, 24855, 24856, 24857, 24858, 24859, 24860, 24861, 24862, 24863, 24864, 24865, 24866, 82208, 82209, 82210, 21518, 21431, 21432, 21433, 21434, 21435, 21436, 21437, 21438, 21439, 21440, 21441, 21442, 21443, 21444, 21445, 21446, 21447, 21448, 21449, 21450, 21451, 621, 21452, 21453, 21454, 21495, 21496, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549]

Question 11

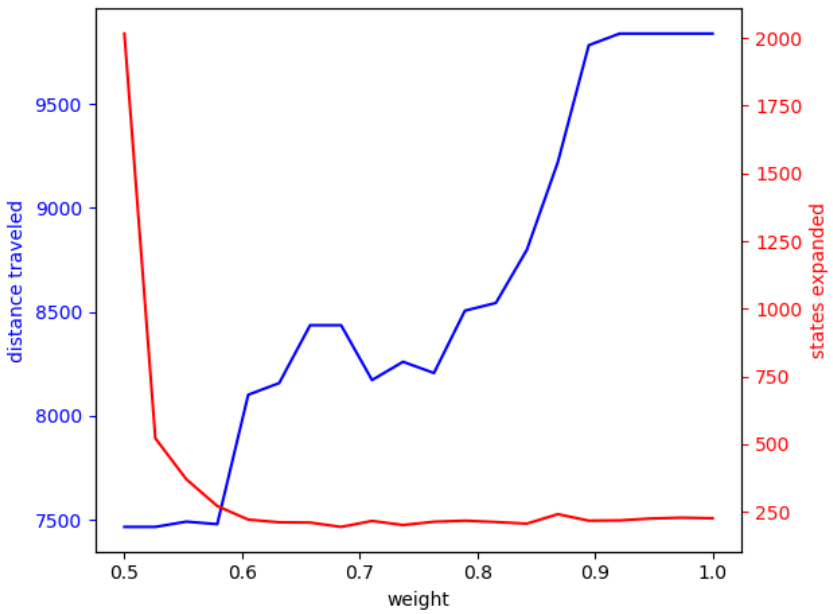
load_map_from_csv: 1.39sec

Solve the map problem. Map(src: 54 dst: 549) A* (h=AirDist, w=0.500) time: 0.09 #dev: 2016 total_cost: 7465.52897 |path|: 137 path: [54, 55, 56, 57, 58, 59, 60, 28893, 14580, 14590, 14591, 14592, 14593, 81892, 25814, 81, 26236, 26234, 1188, 33068, 33069, 33070, 15474, 33071, 5020, 21699, 33072, 33073, 33074, 16203, 9847, 9848, 9849, 9850, 9851, 335, 9852, 82906, 82907, 82908, 82909, 95454, 96539, 72369, 94627, 38553, 72367, 29007, 94632, 96540, 9269, 82890, 29049, 29026, 82682, 71897, 83380, 96541, 82904, 96542, 96543, 96544, 96545, 96546, 96547, 82911, 82928, 24841, 24842, 24843, 5215, 24844, 9274, 24845, 24846, 24847, 24848, 24849, 24850, 24851, 24852, 24853, 24854, 24855, 24856, 24857, 24858, 24859, 24860, 24861, 24862, 24863, 24864, 24865, 24866, 82208, 82209, 82210, 21518, 21431, 21432, 21433, 21434, 21435, 21436, 21437, 21438, 21439, 21440, 21441, 21442, 21443, 21444, 21445, 21446, 21447, 21448, 21449, 21450, 21451, 621, 21452, 21453, 21454, 21495, 21496, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549]

Question 12

When the weight is 0.5, the algorithm is exactly the A* algorithm. As the weight increases, the algorithm gets closer to the greedy best search algorithm, with a weight value of 1 being the greedy best search algorithm itself.

As the weight increases, the algorithm relies more and more on the heuristic function and less and less on the cost of the current path, as we've seen in class. In addition, as the weight increases, we can see that the computation becomes computationally easier as less nodes are being expanded.



Question 14

The MaxAirDist heuristic is indeed admissable, since it does not take into account all other nodes that need to be visited except for the node with the maximal air distance from the current node that has not been visited yet.

In other words, if there is only one more node that needs to be visited before a goal state is reached, the MaxAirDist heuristic will return the exact distance to the goal state, which is h^*, else, it will return a smaller value than h^*, hence it is indeed admissable.

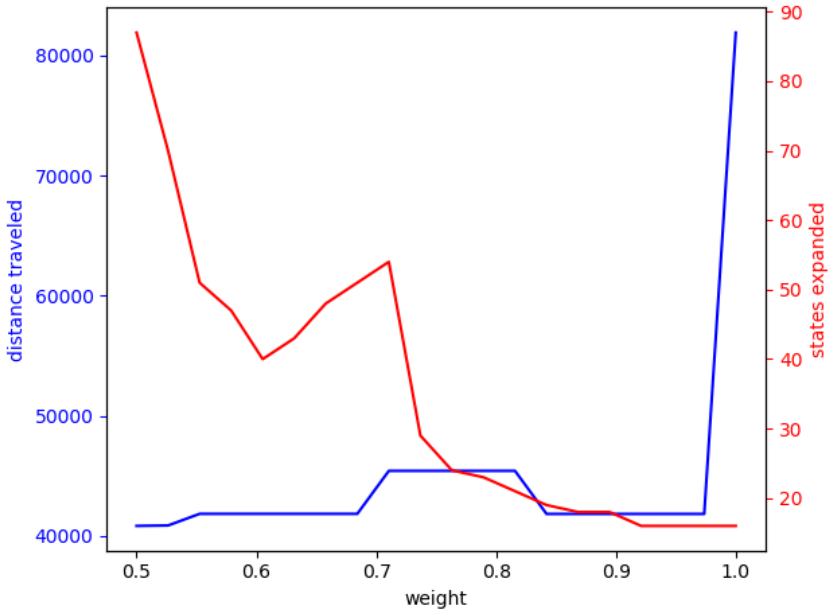
Question 16

Solve the relaxed deliveries problem. RelaxedDeliveries(big_delivery) A* (h=MaxAirDist, w=0.500) time: 3.82 #dev: 3908 total_cost: 40844.21165 |path|: 11 path: [33919, 18409, 77726, 26690, 31221, 63050, 84034, 60664, 70557, 94941, 31008] gas-stations: [31221, 70557]

Question 17

Solve the relaxed deliveries problem. RelaxedDeliveries(big_delivery) A* (h=MSTAirDist, w=0.500) time: 1.19 #dev: 87 total_cost: 40844.21165 |path|: 11 path: [33919, 18409, 77726, 26690, 31221, 63050, 84034, 60664, 70557, 94941, 31008] gas-stations: [31221, 70557]

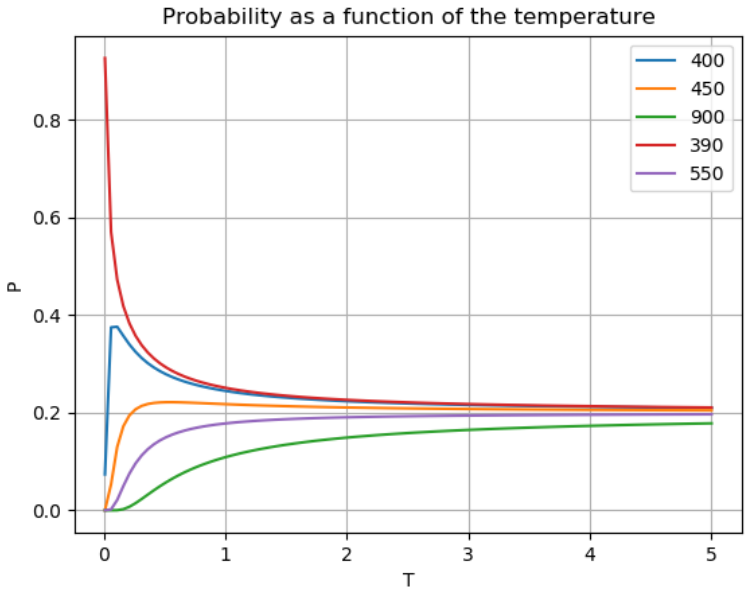
Question 18



Question 19

$$\forall x_i \in x^t: \quad \Pr(x_i) = \frac{\left(\frac{x_i}{\alpha}\right)^{-1/T}}{\sum_j \left(\frac{x_j}{\alpha}\right)^{-1/T}} = \frac{x_i^{-1/T} \cdot \alpha^{1/T}}{\sum_j \left(x_j^{-1/T} \cdot \alpha^{1/T}\right)} = \frac{x_i^{-1/T} \cdot \alpha^{1/T}}{\alpha^{1/T} \sum_j x_j^{-1/T}} = \frac{x_i^{-1/T}}{\sum_j x_j^{-1/T}}$$

Question 20



Question 21

First let us notice that the expression can be rewritten in the form:

$$\forall x_i \in x: \quad \Pr(x_i) = \frac{x_i^{-1/T}}{\sum_{j\in[N]} x_j^{-1/T}} = \cdots = \frac{1}{1+\sum_{i\neq j}\left(\frac{x_i}{x_j}\right)^{1/T}}$$

And by taking the limit T\to0, we have two options:

1. if $x_i < x_j$ for every $i \neq j$ then $\sum_{i\neq j}\left(\frac{x_i}{x_j}\right)^{1/T} \xrightarrow[T\to 0]{} 0$ and $\lim_{T\to 0}\Pr(x_i) = \frac{1}{1+0} = 1$. This is the case where $x_i = \min_j \{x_j\}_{j=1}^N = \alpha$ is the minimal element.

2. if there exists at least one j such that $x_i \geq x_j$ then we get:

   (a) $\left(\frac{x_i}{x_j}\right)^{1/T} \xrightarrow[T\to 0]{} \infty$ if $x_i > x_j$, which leads to $\lim_{T\to 0}\Pr(x_i) = \infty$. This is the case where x_i is not the minimal element of $\{x_j\}_{j=1}^N$.

   (b) $\left(\frac{x_i}{x_j}\right)^{1/T} = (1)^{1/T} = 1 \xrightarrow[T\to 0]{} 1$, if there is exactly one j that satisfies the equality $x_i = x_j$.

   (c) If there are $\{x_k\}_{k\in K}$ where $K \subseteq N$ and $2 \leq |K| \leq |N|$ that satisfy $x_i = x_k$ for all $k \in K$, and all other coordinates satisfy $x_i < x_j$ (such that their summands vanish), we get $\lim_{T\to 0}\Pr(x_i) = \frac{1}{1+|K|\cdot 1+(0+\cdots+0)} = \frac{1}{1+K}$.

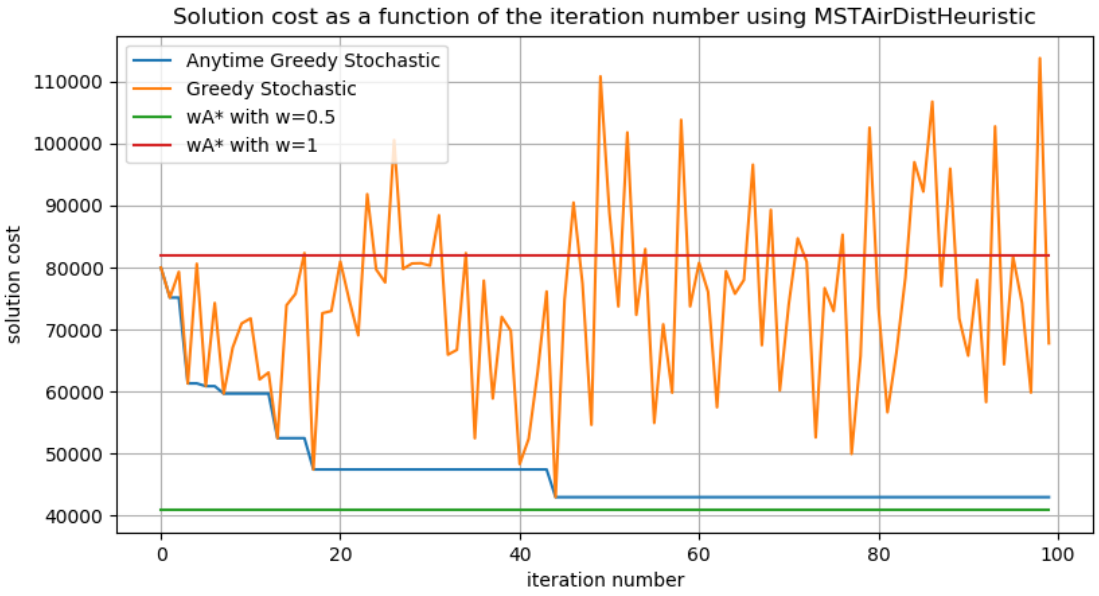
Question 22

By using the equivalent expression of the probability function from question 21 and taking the limit $T \to \infty$ while assuming $x_j \neq 0$ for all j, we get

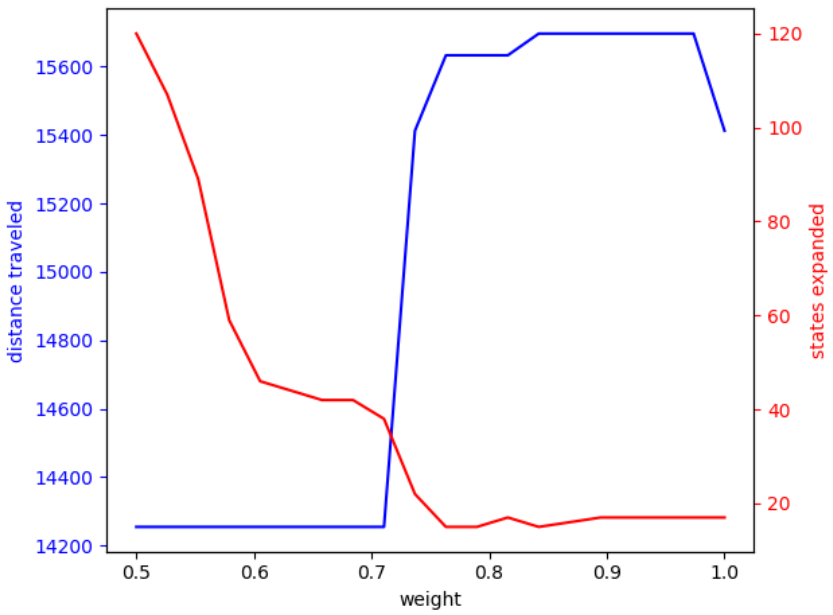
$$\lim_{T\to\infty} \Pr(x_i) = \lim_{T\to\infty} \frac{1}{1+\sum_{i\neq j}\left(\underbrace{\frac{x_i}{x_j}}_{\neq 0}\right)^{1/T}} = \frac{1}{1+\underbrace{(1+\cdots+1)}_{N-1 \text{ times}}} = \frac{1}{N}$$

and since in our case $N = 5$, we get that indeed the limit is $\frac{1}{5} = 0.2$, as can be seen in the plot.

Question 24



Question 26



Question 27

Let us consider the following Relaxed Deliveries Heuristic function h:

For a state s, define the heuristic value $h(s)$ as the cost of the solution found by the Relaxed Deliveries Problem with the initial state as the current state the heuristic function received as input, and the final state as the current problem's final state. The gas stations, drop points, current fuel level and full-tank fuel level are all the same as in the original problem. If the Relaxed Deliveries problem did not find a solution from the current state to the goal state, let h return ∞, or equivalently, some very big number that represents a very high, undesireable heuristic value that the algorithm would rather not choose.

h is an admissable heuristic, since its return values always represent a solution whose total cost is comprised of aerial distances, which are always lower than the true costs the scooter must pay to reach the goal state. If h returns ∞, then necessarily there is no valid path for the strict problem as well, so we might as well treat its cost as ∞, since we cannot actually get to a goal state.

Question 28

The results we got are:

StrictDeliveries(small_delivery) A* (h=RelaxedProb, w=0.500) time: 16.68 #dev: 80 total_cost: 14254.79234 |path|: 8 path: [43516, 67260, 17719, 43454, 43217, 32863, 7873, 42607] gas-stations: [17719, 32863]

When comparing the result to question 26, we get that for $w = 0.5$, the total solution cost is the same, but the number of total states expanded is better by 33.33% (120 vs. 80).

for $w \geq 0.58$ (approximately) we get that the algorithm from question 26 is superior in terms of states expanded. Taking such a w will result in a total cost increase of about 600.

As for the the runtime, the algorithm from question 26 for $w = 0.58$ took 11.06 seconds to finish. The algorithm in this question took 16.68 seconds. This is a substantial runtime setback, as it is worse by about 50.81% relative to question 26.

Theoretical Question

1. From the fact that h is admissible we know that $\forall s \in S$ such that $\text{Applicable}_h(s)$ is True , $h_0(h,s) = h(s) \leq h^*(s)$. In addition, $\forall s \in S$ such that $\text{Applicable}_h(s)$ is False , $h_0(h,s) = 0 \leq h^*(s)$ because the price function is bounded from below by $\delta > 0$ so $h^*(s) \geq 0, \forall s \in S$. From that we conclude that h_0 is also admissible.

2. We assume that the state space is a tree. Our suggested Heuristic is:

$$h'(v) = \begin{cases} h(v) & \text{if } \text{Applicable}_h(v) \text{ is true} \\ 0 & \text{if } \text{Applicable}_h(v) \text{ is false} \wedge \text{isGoal}(v) \text{ is true} \\ \min_{u \in \text{Succ}(v)} (\text{cost}(v,u)) & \text{otherwise} \end{cases}$$

   If $\text{Applicable}_h(v)$ is True then $h'(v) \leq h^*(v)$ because h is admissible and if isGoal(v) is True and $\text{Applicable}_h(v)$ is False then $h'(v) = 0 \leq h^*(v)$ because the price function is positive. Otherwise, from the fact that the price function is positive we know the minimal cost of a path from v to a goal state is at least the minimal cost of an edge from v to one of its successors, therefore, $h'(v) \leq h^*(v)$ and it means h' is admissible as required. Now, for each v in the tree $h_0(v) \leq h'(v)$ because if $\text{Applicable}_h(v)$ is True or if isGoal(v) is True then $h_0(v) = h'(v)$ and otherwise $h_0(v) = 0 < \delta \leq h'(v)$, h' is more informed than h_0.

3. As can be seen in section b, the fact that the state space is a tree was not actually used, therefore the same heuristic function h' can be used on any state space.
   Another solution could be to add information to each state s in our state space; whether it has been visited already or not. If it hasn't been visited its heuristic value will be $h'(s)$ (as in section b). If it has been visited then the heuristic value that will be returned is $h_0(h,s)$. Since for every state s, $h'(s) \geq h_0(h,s)$ and clearly $h_0(h,s) \geq h_0(h,s)$, in addition to what was defined and explained in section b, the heuristic function we've suggested is this section is indeed more informed than h_0, and still admissible

4. The claim is true.
   Let us first note that the A* algorithm with the given $h_0(h',s)$ heuristic actually behaves like the Uniform Cost algorithm, since the heuristic value of all nodes is zero (as in Uniform Cost) except the initial node, and the initial node's heuristic value doesn't actually affect the algorithm's behaviour since the initial node does not 'compete' with any other node before being expanded.
   From the fact that $h'(s_0)$ is given (s_0 is the initial state) and equals $h^*(s_0)$, we can use a variation of the DFS-L algorithm, which replaces the depth restriction input with a cost restriction instead, and use it with a restriction equal to $h'(s_0)$. the algorithm in this case is admissible because it is given in the question that there exists a solution with a cost of $h'(s_0)$ and it is optimal, hence the algorithm will necessarily find such a solution, as we've learned for DFS-L.