Artificial Intelligence

Homework #1

Alexander Shender

328626114

Idan Finkelstein

208686121

1.

We calculate the number of permutations as a factorial, since we cannot visit the same place twice.

K	#possiblePaths	Log2(possiblePaths)	Calculation time
10	3628800	21.79	< 1 [s]
13	6227020800	32.53	5.799 [s]
15	1307674368000	40.25	20.298 [min]
16	20922789888000	44.25	5.41 [hr]
17	355687428096000	48.33	3.834 [day]
20	2432902008176640000	61.07	71.84 [year]
21	51090942171709440000	65.46	1508.82 [year]
24	620448401733239439360000	79.03	18.323 [mil. Year]

2.

The *maximum* outdegree exists in the initial node v_0, where no apartment or lab were visited yet. Then, an agent can visit any of those locations using the corresponding operation. Thus:

$$outdegree_{\max} = \sum Labs + \sum Apartments$$

The *minimum* outdegree can be 1, in the following cases:

- Ambulance is at any apartment, and only 1 lab available in the city
- Ambulance is at apartment, there is no room in the refrigerator on ambulance, and there is only 1 laboratory in the city.
- Ambulance is at the lab, and only 1 apartment left to visit.

3.

Circles are not possible, out of constrains that have been put here:

- After visiting an apartment, an agent cannot return there
- To visit a lab, the refrigerator has to be not empty or it has to increase the number of matoshim. Thus, the agent cannot infinitely visit labs, since it will take all the matoshim from them eventually and will have to visit an apartment between visiting the labs. And will eventually reach the goal state.

4.

First, not all the states are achievable. Assume a problem is given where no solution can be found (number of roommates in all apartments is bigger than total number of matoshim). Then, the goal state is not achievable. Or a state with a number of matoshim, which cannot be achieved even by collecting all the matoshim in the city.

To define the total number of states, we need to define the maximum number of matoshim available in the city (otherwise, no other limit is given to this state parameter):

$$total\ matoshim\ num = \sum_{l_i \in LABS}^{LABS} l_i.matoshim$$

The number of optional locations:

$$locations\ num = \sum LABS + \sum APARTMENTS$$

The optional number of visited labs:

$$visited\ labs\ num = \sum LABS$$

The number of permutations that a state can have with the $Taken$ amd $Transferred$ Groups, given that they are under constraint $Taken \cup Transferred \in \emptyset$

For example, if total Apartments number is 7, and $\sum Taken \cap Transferred = 7$, we can write $Taken \cap Transferred$ as $\{(7,0), (6,1) \dots (0,7)\}$. If $\sum Taken \cap Transferred = 6$, it can be $\{(6,0), (5,1) \dots (0,6)\}$. Total number of such permutations for each case is

$\sum Taken \cap Transferred + 1$. Thus, the number of such parameters permutations is:

$$taken\ \&\ transfered\ num = \sum_{i=0}^{Apartments\ num} (i+1)$$

Thus, the total number of states:

$$\begin{aligned} TOTAL\ STATES\ NUM &= total\ matoshim\ num \cdot\ locations\ num \cdot visited\ labs\ num \\ &\cdot taken\ \&\ transfered\ num \end{aligned}$$

5.

Under the current constrains, if the solution exists, there are no sinks. Sink can exist, if for example, the number of matoshim available in the city is less than the number of total roommates in apartments. Then, the condition to visit some apartments will never be satisfied, and the goal state never reached.

6.

It depends on the initial state. If the ambulance has a sufficient initial number of matoshim to collect all tests, it will have to visit all apartment and visit a lab just once in the end.

$$shortest\ path\ possible = 1 + \sum APARTMENTS$$

In the longest path, when there are 0 matoshim in the initial state, and the agent will visit lab after each apartment visit, the path length is:

$$longest\ path\ possible = 1 + 2 \cdot \sum APARTMENTS$$

7.

Defining the successors. The ‘canVisit’ conditions are defined in the assignment. The ‘i’ index refers to all the possible Labs or Apartments.

$$Succ(s) = \left\{ \begin{matrix} l_i.loc, \\ Taken = \emptyset, \\ Transfered = s.transfered + s.taken, \\ matoshim = s.matoshim + l_i.matoshim, \\ visited\ labs = s.visited\ labs \cap l_i \end{matrix} \middle| canVisit(s, l_i) \right\}$$
$$\cup \left\{ \begin{matrix} d_i.loc \in Apartments.loc, \\ Taken = s.taken + d_i.roommates, \\ Transferred = s.transferred, \\ matoshim = s.matoshim - d_i.roommates, \\ visited\ labs = s.visited\ labs \end{matrix} \middle| canVisit(s, d_i) \right\}$$

8.

The output is:

```
StreetsMap(src: 54 dst: 549)        UniformCost              time:   0.73   #dev: 17354   |space|: 17514    total_g_cost:  7465.52560
|path|: 136   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]
```

9.

wet

10.

wet

11.

A^* with NULL heuristic returned:

```
time:  0.80  #dev: 17354  |space|: 17514  total_g_cost: 7465.52560  |path|: 136
```

A^* with AirDist heuristic returned:

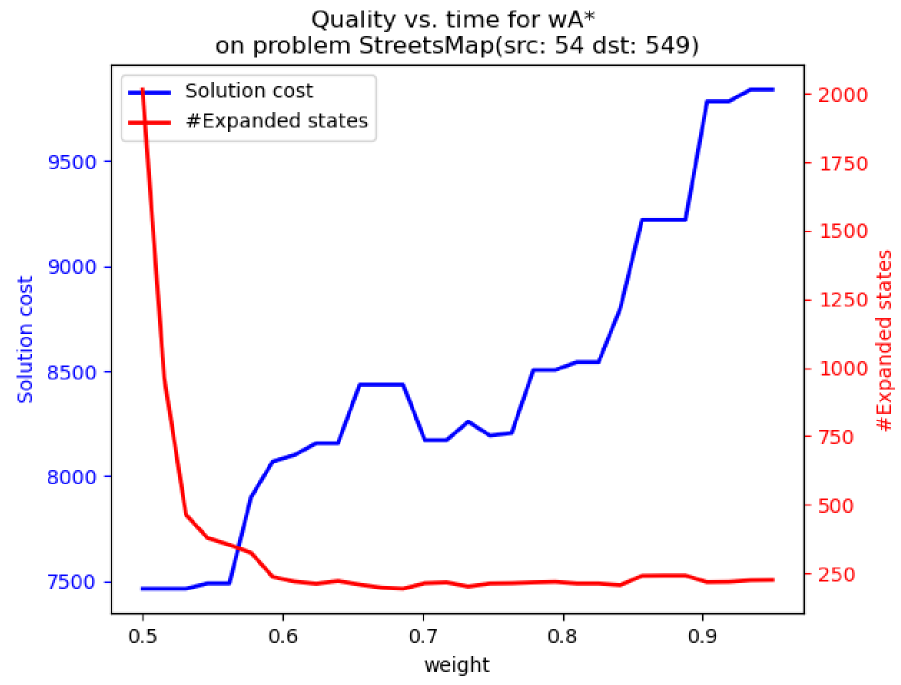
```
StreetsMap(src: 54 dst: 549)      A* (h=AirDist, w=0.500)     time:  0.17  #dev: 2015   |space|: 2229   total_g_cost:
7465.52560  |path|: 136  path: [  54 ==>   55 ==>   56 ==>   57 ==>   58 ==>   59 ==>   60 ==> 28893 ==> 14580 ==>
14590 ==> 14591 ==> 14592 ==> 14593 ==> 81892 ==> 25814 ==>   81 ==> 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]
```

We can see that same path was found, but in much shorter time, with much less space and expanded nodes amount.

We have developed $\frac{17354 - 2015}{17354} = 88\%$ less nodes.

12.

The following graph was received after the code was completed and run successfully:



We can indeed see the general tendency, that higher weights give a solution with higher cost, but with less expanded states. But we can see that for $w > \sim 0.55$, the increase in the w parameter does increase the solution cost, but barely saves (and even increases, as in range $\sim 0.85 < w < \sim 0.9$) a number of expanded states.

The optimal choice would be to take a value of w in the range $[0.55, 0.6]$.

13.

wet

14.

wet

15.

(i) The line of code (or specifically, frozen = True) which makes the class immutable is the decorator:

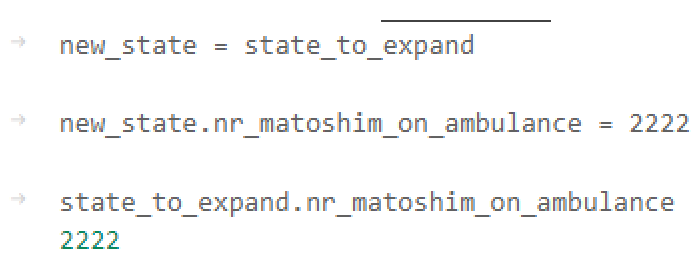
```
@dataclass(frozen=True)
```

(ii) As written in the python docs on the internet:

It is not possible to create truly immutable Python objects. However, by passing `frozen=True` to the dataclass() decorator you can emulate immutability. In that case, dataclasses will add __setattr__() and __delattr__() methods to the class. These methods will raise a FrozenInstanceError when invoked.

Additional way to ensure the object is immutable is making its fields (all except of "nr_matoshim_on_ambulance") an object of type Frozenset. This ensures nothing can be added / deleted from those lists

(iii) If frozen parameter would be False, a user may have mistakenly copied the reference to the "state_to_expand", (thinking he's copying an object) and make changes to it. This will result in the "state_to_expand" changes. Example:



```
→ new_state = state_to_expand

→ new_state.nr_matoshim_on_ambulance = 2222

→ state_to_expand.nr_matoshim_on_ambulance
  2222
```

16.

wet

17.

wet

18.

The definition of the admissible heuristic $h(s)$ is:

$$0 \leq h(s) \leq h^*(s)$$

Where the $h^*(s)$ is the optimal heuristic.

The MaxAirDistHeuristic **is admissible.** Proof: it returns the max Euclidean distance between the points. The only way the actual path length will be equal to this distance is when ALL of those points are on the same line, and the order in which we visit them is from the closest to the furthest. In this case, $h(s)$ will be equal to $h^*(s)$. Other than that (any other case), the actual optimal distance we'll have to travel is bigger than MaxAirDistHeuristic.

19.

wet

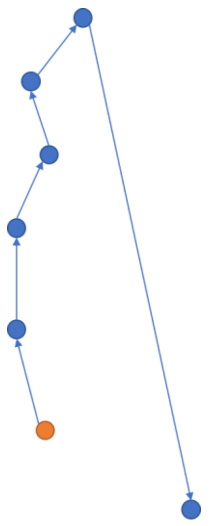
20.

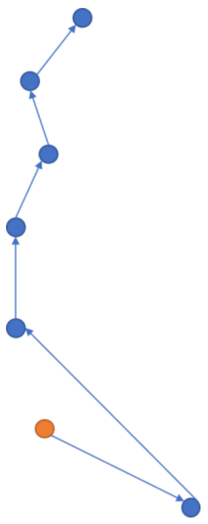
wet

21.

This heuristic **is not admissible**! Following the closest node may give us the path which isn't the most optimal one. It's easier to illustrate an example to show where $h(s) > h^*(s)$:

$h(s)$ would give us the following path on the left, and it's distance would be greater than the distance of $h^*(s)$, which is on the right:





22.

wet

23.

wet

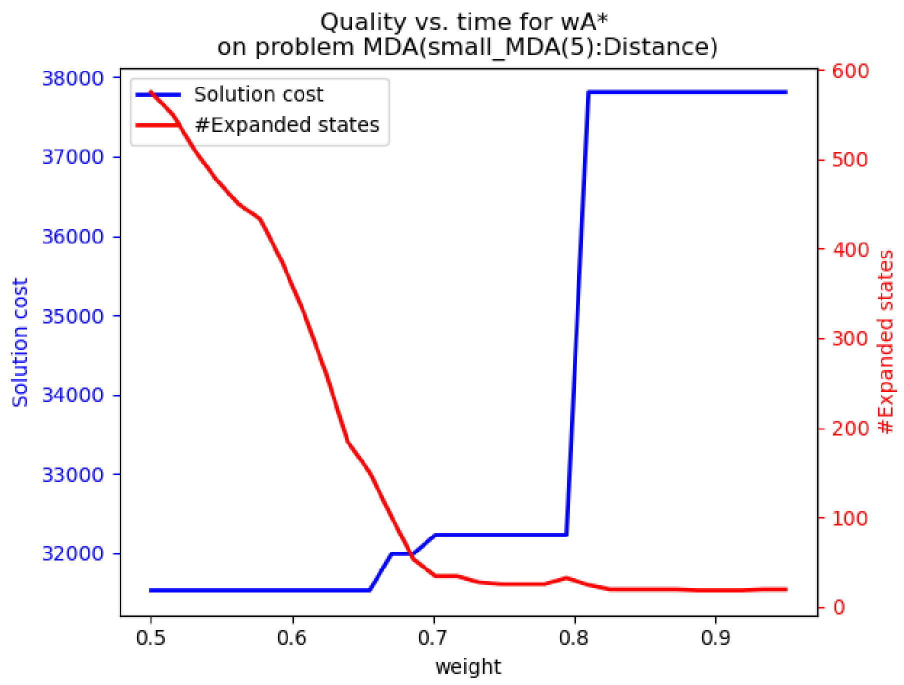
24.

MST heuristics measures the minimum spanning tree weight, using the distance cost. The MST will find the shortest path connecting all the nodes in the graph. It cannot give a cost which is greater than other spanning tree. Thus, this heuristic **is admissible**.

25.

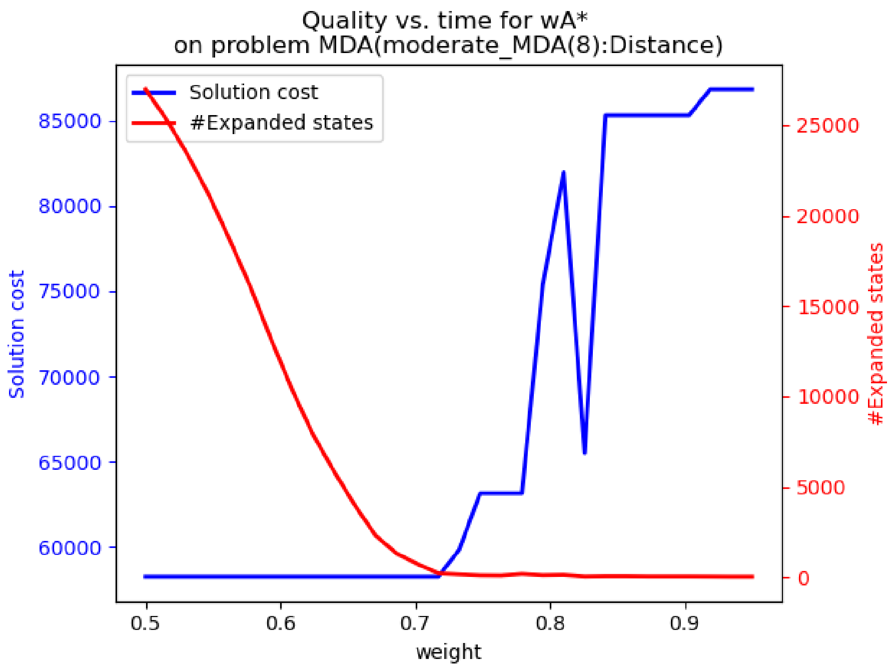
The resulting graphs are the following:

For a small MDA problem, with MST Air distance heuristic:



We can observe that for w less than 0.7, the solution cost has very to no changes. And that for $w > 0.8$, the solution cost increases dramatically at no decrease in the number of expanded states. The optimal value would lie in the range: $\sim 0.7 < w < 0.8$

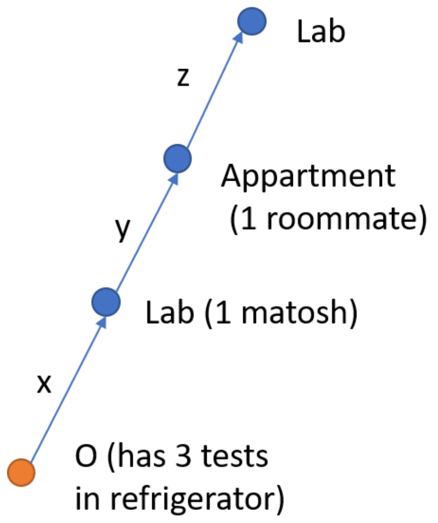
For a medium MDA problem, the following graph was obtained:



In the same way, increasing w from 0.5 to ~ 0.72 - there is no improvement in solution cost at all, but a huge decrease in the number of expanded states. And increasing the w from ~ 0.72 towards 1 will give almost none (or none) benefit in the number of expanded states, but increase the solution cost substantially. Thus, the optimal choice here would lay in the region very close to $w = \sim 0.72$, or the point seen as the pit in the image.

26.

MDAMaxAirDistHeuristic with $cost_{MDA}^{test\ travel}$ cost function is NOT admissible. The easiest way it to visualize it. We take the same example as with the regular cost function $cost_{MDA}^{dist}$ where all the remaining locations to visit are **on the same line**. Distances marked as x, y, z. O is the initial location. We can see that visiting the nodes with the arrows we get the optimal path:



It is obvious that $cost_{MDA}^{dist} = (x + y + z)$, the maximum distance between points

In this case, the MDAMaxAirDistHeuristic heuristic will result in: $h(s) = 3 \cdot (x + y + z)$

While the optimal heuristic is: $h^*(s) = 3 \cdot x + 1 \cdot z = 3x + z$

Which proves: $h(s) > h^*(s)$, thus the heuristic is **not admissible**.

27.

This heuristic was not admissible with the regular cost. For the same reason it will **not be admissible** now. In fact, we can use same example as in paragraph 26, where the MDASumAirDistHeuristic will result in same value as MDAMaxAirDistHeuristic, and both are bigger than $h^*(s)$.

28.

Also MDAMSTAirDistHeuristic **is not admissible**. By taking the same example from paragraph 26. It does find the shortest path connecting all nodes. But it doesn't take into account that we can drop tests at the lab to decrease this cost for some of the path.

29.

wet

30.

MDATestsTravelTimeToNearestLabHeuristic **is admissible.** The most optimal solution with regards to the $cost_d^{tests\ travel}$ would be to bring the tests, which are currently on the ambulance, to a closest lab, and to bring the tests from each apartment to the closest laboratory around. This is exactly what this heuristic estimates. Thus is satisfies the condition for the admissible heuristic:

$$0 \leq h(s) \leq h^*(s)$$

31.

The output from running the Moderate MDA problem with MDATestsTravelTimeToNearestLabHeuristic are the following:

```
MDA(moderate_MDA(8):TestsTravelDistance)   A* (h=MDA-TimeObjectiveSumOfMinAirDistFromLab, w=0.500)

time:  57.93   #dev: 28853   |space|: 41799    total_g_cost: 104387.48471

total_cost: MDACost(dist= 148981.147m, tests-travel= 104387.485m)   |path|: 19
```

Comparing to the solutions found to the same problem with other Heuristics, which relied on the $cost_d^{dist}$:

- MDAMaxAirDistHeuristic :
  total_cost: MDACost(dist= 58254.187m, tests-travel= 131811.935m) |path|: 13
- MDASumAirDistHeuristic :
  total_cost: MDACost(dist= 58254.187m, tests-travel= 131811.935m) |path|: 13
- MDAMSTAirDistHeuristic:
  total_cost: MDACost(dist= 58254.187m, tests-travel= 131811.935m) |path|: 13

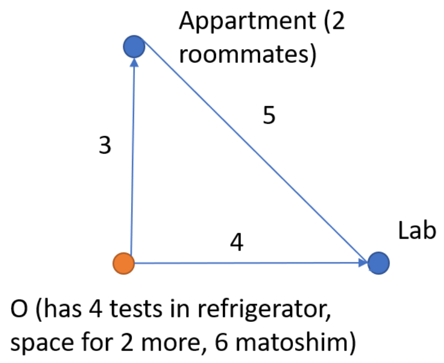
We can see that indeed, the tests-travel distance **has decreased.** The ‘dist’ cost has increased, but this isn’t our optimization objective with MDATestsTravelTimeToNearestLabHeuristic. The ‘path’ – number of paths made, has also increased – the agent visits the lab more often to drop the tests.

32.

The algorithm is **complete**. In general, the UCS algorithm is complete, if the tree is finite, and if there are no cycles. There will be no cycles as stated earlier in the report due to the limitations of the problem. $A1$ presents a filter on paths, which are longer than the minimum found path with C_{dist} cost function with some margin - $(1+\epsilon)\cdot C^*_{dist})$. But there is still paths which do pass through this filter, at least one of them, which is the path with the C^*_{dist} cost itself. Thus the algorithm will return a solution if such exists.

33.

The algorithm **will not always return the most optimal solution**, as the most optimal solutions in regard to the $cost_{MDA}^{test\ travel}$ cost function may be filtered out in the first stage. Easiest is to visualize. Assume $\epsilon = 0.6$, as in the Ex. 34.


Appartment (2 roommates)
5
3
4
Lab
O (has 4 tests in refrigerator, space for 2 more, 6 matoshim)

Using the UCS, the next node to expand is the closest one. Regarding to O, this is Appartment. The C^*_{dist} in this case will be O-> app -> lab = 3+5 = 8. Visiting the apartment before the lab is obviously the best solution if we take into account only the distance. Then, $(1+\epsilon)\cdot C^*_{dist} = 8 * 1.6 = 12.8$. The second solution (O -> lab -> app -> lab) will cost $cost_{dist} = 4 + 5 + 5 = 14 > C^*_{dist} = 12.8$, thus will be filtered out. Now calculating the $cost_{MDA}^{test\ travel}$ for both paths:

$$cost_{MDA}^{test\ travel}\,(O \rightarrow lab \rightarrow app \rightarrow lab) = 4\cdot 4 + 5\cdot 0 + 5\cdot 2 = 26$$

$$cost_{MDA}^{test\ travel}\,(O \rightarrow app \rightarrow lab) = 4\cdot 3 + 5\cdot 6 = 42$$

Obviously, $cost_{MDA}^{test\ travel}\,(O \rightarrow lab \rightarrow app \rightarrow lab)$ is the optimal path according to $cost_{MDA}^{test\ travel}$ cost function, but it was filtered out in the first stage.

34.

By running the algorithm with the TravelDistance cost function, the optimal algorithm returned was of length $C^*_{dist} = 58254.2$, which resulted in a threshold of $(1+\epsilon) \cdot C^*_{dist} = \ (1+0.6) \cdot C^*_{dist} = 93206.7$

After running the second step of the algorithm, $DistCost(ReturnedSolution) = 89430.7$.

1. It does suit the condition:

$$DistCost(ReturnedSolution) < \ (1+\epsilon) \cdot C^*_{dist}$$

2. The required relation:

$$\frac{C^*_{dist}}{DistCost(ReturnedSolution)} = \frac{58254.2}{89430.7} \approx 0.65$$

Which means this solution is much longer (regarding to pure distance cost function).

Comparing those results to the results from previous runs:

	#dev	|space|	Dist	Dist-Tests
A2	28438	40498	89430.664	104387.485
A* (ex 31)	28853	41799	148981.147	104387.485
A* (ex 23)	29766	40024	58254.187	131811.935

We can observe that we have received **the same** tests-distance as in the case in Ex.31, where only MDATestsTravelTimeToNearestLabHeuristic heuristic was used, but the physical distance than the ambulance had to travel had decreased dramatically.

35.

The statement is incorrect. This can be seen by trying small values of ϵ in code, where the algorithm won't return any solution for small values of it.

Theoretical prove: The cost of the nodes that are placed in the OPEN queue can be updated in the A* algorithm, if the new cost for reaching those nodes is smaller than the existing cost. If for example some solution passes through node B, and the first cost reaching B found it greater than $(1+\epsilon) \cdot C^*_{dist}$, this node will not enter the OPEN queue. Thus it means that automatically we don't develop this solution.

This cost may be higher than $(1+\epsilon) \cdot C^*_{dist}$, but would later be improved by discovering new, cheaper cost. But since we don't put this node in OPEN, we cancel this option, and this solution will be not be developed. Thus, some solutions may be lost.

36.

The statement is correct under the combined criterion. It was shown earlier that MDATestsTravelTimeToNearestLabHeuristic is admissible, thus with the A* algorithm, it will return the optimal solution from the $DistEstOptimal$ group.

37.

Several benefits of A2 over A2:

1. It does not store nodes in the Open queue, the cost of which does not meet the condition of being less than $(1+\epsilon)\cdot C^*_{dist}$
2. It runs the UCS search algorithm, which always choses to open the next closest node, and doesn't relate to any heuristics.

38.

wet

39.

The reference from the regular A* :

```
MDA(small_MDA(5):Distance)          A* (h=MDA-MST-AirDist, w=0.500)   time:  13.63   #dev: 575
|space|: 947      total_g_cost: 31528.65909   total_cost: MDACost(dist=  31528.659m, tests-travel=
52112.429m)   |path|: 8
```

And output using the E* epsilon:

```
MDA(small_MDA(5):Distance)          A*eps (h=MDA-MST-AirDist, w=0.500)

time:  15.46   #dev: 564     |space|: 933      total_g_cost: 31528.65909

total_cost: MDACost(dist=  31528.659m, tests-travel=  52112.429m)   |path|: 8
```

We have developed less states (564 < 575) as expected, since we use the Air dist SUM heuristic, allowing the solution to be ϵ times bigger than the optimal soluition.

40.

Wet

41.

The result obtained is the following:

```
MDA(moderate_MDA(8):Distance)      Anytime-A* (h=MDA-MST-AirDist, w=0.800)
time:  37.76   #dev: 1027    |space|: 740     total_g_cost: 64055.65000
total_cost: MDACost(dist=  64055.650m, tests-travel= 131870.337m)   |path|: 13
```

We compare the results with the regular A* with the same heuristic function MDAMSTAirDistHeuristic from Ex 23:

	#dev	|space|	Dist	Dist-Tests
Anyitime A*	1027	740	64055.650	131870.337
A* (ex 23)	29766	40024	58254.187	131811.935

We can indeed observe that under the given constrains of max 150 number states to expand the best solution was found for $w = 0.8$. We can see that much less states were developed, and the amount of space (storage) required for this run is much smaller. But the distance is larger (Dist, Dist-tests) than without this constrain – we get a less quality solution.