## 15.

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

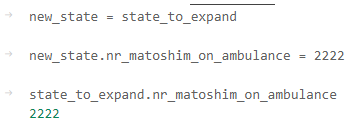
@dataclass(frozen=True)

1. 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()](https://docs.python.org/3/library/dataclasses.html#dataclasses.dataclass) decorator you can emulate immutability. In that case, dataclasses will add [\_\_setattr\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__setattr__) and [\_\_delattr\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__delattr__) methods to the class. These methods will raise a [FrozenInstanceError](https://docs.python.org/3/library/dataclasses.html#dataclasses.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

1. 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:



## 18.

The definition of the admissible heuristic is:

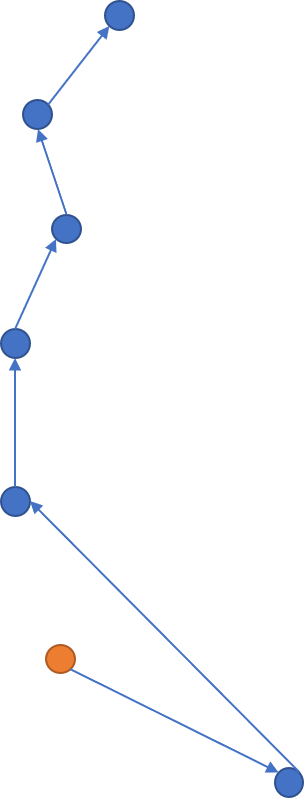
Where the is the optimal heuristic.

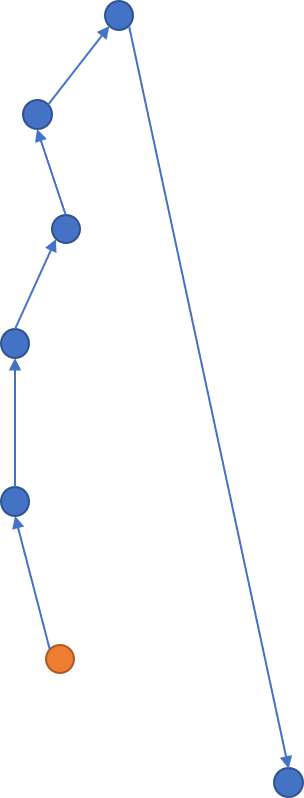
The MaxAirDistHeuristic **is admissible.** Proof: it returns the max Euclidean distance between the points. The only way the actual path length will be equal to this distance is when ALL of those points are on the same line, and the order in which we visit them is from the closest to the furthest. In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

## 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 :

would give us the following path on the left, and it’s distance would be greater than the distance of , which is on the right:



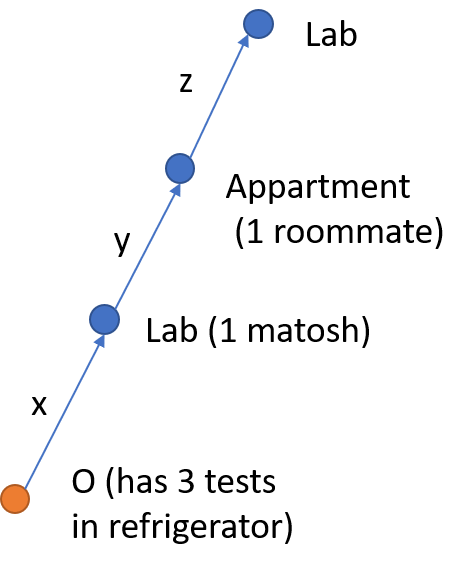


## 24.

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**.

## 26.

MDAMaxAirDistHeuristic with cost function is NOT admissible. The easiest way it to visualize it. We take the same example as with the regular cost function 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 , the maximum distance between points

In this case, the MDAMaxAirDistHeuristic heuristic will result in:

While the optimal heuristic is:

Which proves:, 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 .

## 28.

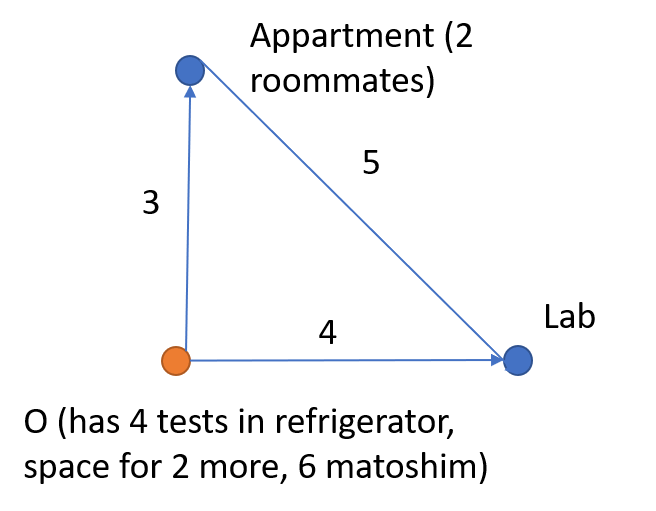
Also MDAMSTAirDistHeuristic **is not admissible**. By taking the same example from paragraph 26. It does find the shortest path connecting all nodes. But it doesn’t take into account that we can drop tests at the lab to decrease this cost for some of the path. The possible solution would be to estimate the tests travel distance to the nearest lab, as is proposed in MDATestsTravelTimeToNearestLabHeuristic…

## 32.

The algorithm is **complete**. In general, the UCS algorithm is complete. presents a filter on paths, which are longer than the minimum found path with cost function with some margin - . But there is still paths which do pass through this filter, at least one of them, which is the path with the 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 function may be filtered out in the first stage. Easiest is to visualize. Assume , as in the Ex. 34.



Using the UCS, the next node to expand is the closest one. Regarding to O, this is Appartment. The 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, . The second solution (O -> lab -> app -> lab) will cost , thus will be filtered out. Now calculating the for both paths:

Obviously, is the optimal path according to 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 , which resulted in a threshold of

After running the second step of the algorithm, .

1. It does suit the condition:
2. The required relation:

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

## 35.

## 36.

## 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
2. It runs the UCS search algorithm, which always choses to open the next closest node, and doesn’t relate to any heuristics.

## 39.

Reference:

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

MDA(small\_MDA(5):Distance) A\*eps (h=MDA-MST-AirDist, w=0.500) time: 132.45 #dev: 846 |space|: 536 total\_g\_cost: 31528.65909 total\_cost: MDACost(dist= 31528.659m, tests-travel= 52112.429m) |path|: 8

Same solution.

More opened nodes, less space..