Home Work 1- Question 2

group name: rajthakk-nabiyuni-bhsinha-a1

Sep. 2018

Question 2

To solve this problem first we formulated the problem in this manner:

a) Set of Valid states

All available cities in the map would be our valid states. Successor function will return a subset of this set at any time.

b) Initial State

The initial state of the map is a city that the path would start from that point.

c) Successor function

This function is a function that would return all neighbors of a given city. These neighbors are all the possible states we can go from a given city.

d) Cost function/edge weights

Cost function can has three possible values. *Distance* would be the distance in miles between two cities. In this case the distance values from road-segment file would be considered as edge weights. *Time* is another possible cost function which will be the traveling time from one city to another one and is the division of distance by the speed limit between cities. *Segments* are number of cities that we path and in this case edge weight for each pair of cities would be considered a constant value(e.g. 1).

e) Goal state

The goal state is the end-city that we want to arrive. This city is the last city at the end of our path.

f) Heuristic function

This function is just get calculated for astar routing algorithm and is the distance between current city and the end-city that we want to reach. This distance is calculated by using latitude and longitude of current city and end-city. As it is mentioned in the *assumption* section, for those cities which are not in the city-gps file, a high priority(i.e. a low heuristic function) will be considered to make sure that they would be considered in the process of finding the best path. In other words, as our best path may go through one of these cities without gps values,

we always considered them and remove them first from the fringe to make sure we are not missing any possible candidate for the best route.
We tested three different heuristic functions and we used the first one which is admissible for distance:
1. Real distance in miles: By using Haversine formula we can calculate the distance of two cities in miles. This function is admissible if the cost function is distance, because the closest distance between two cities is the distance that directly connect them to each-other. So this function always underestimates the distance between two cities. In other words, all the paths would have equal or larger distance than the result of the function. So we can say that astar search for distance cost function is optimal.
For any other cost function, this function is not admissible and as a result the result the astar search is not optimal. The reason is that this function calculate the cost and not time or number of segments. We may argue that it would be hard or so expensive to have a admissible function for time. The reason is that we may get closer to a city but the path takes more time than another path which is longer because of the speed limits. In other words the travel time is not consistent across the cities, so it would be hard to find a admissible function for that. A similar argument applies for segments cost function.
2. Manhattan distance in degrees: by using differences of latitude and longitude of the two cities in degrees, we can calculate distance based on degrees instead of miles. This function is not admissible for non of the cost functions because the different between angle of two cities(i.e. latitude) are larger near the equator and smaller near the pole.
3. Euclidean Distance: by using differences of latitude and longitude of the two cities in degrees, we can calculate distance based on degrees instead of miles. This function is not admissible for non of the cost functions because the different between angle of two cities(i.e. latitude) are larger near the equator and smaller near the pole.

How the search algorithm works:
For *bfs, dfs* and *ids* routing algorithms, the program uses a function named *blind_search* which adds all possible successors to a fringe dataset and start poping them based of the type of routing algorithm (i.e. queue for *bfs* and stack for *dfs*).*ids* search would call dfs search and limits its search to a specific depth. The depth would increase after each failure up to the depth 1000. After this depth the program will return *No Path has found!!!* and will terminate the search.
For *uniform* and *astar* routing algorithms, another function named *astar_search* will be called. This function uses a priority queue to simulate best first search and pop a city with the lowest evaluation function value (i.e. *f(s)*). Value of g(s)(which is the first part of f(s)) depends on the type of cost function. It will be the cost of reaching to the current point by considering all costs(edge weights) along the path.
However, the second term of f(s) (which is h(s)) would depend of the type of routing algorithm. For *uniform* search the function assume a zero value for heuristic function and for astar search it will use the first heuristic which was described before.

Problems we faced, any assumptions, simplifications, and/or design decisions:
Assumptions:
- The start-city and end-city are in the city-gps file. In this way we would make sure that our heuristic function would always have a second point for calculation.
- Cities with missing gps in the path have high priority.
- We can skip a revisit city which was visited through a shorter path before. This means that all possible successors of that city considering the shorter path were examined before. So there is no way of missing a possible best path by using this approach.

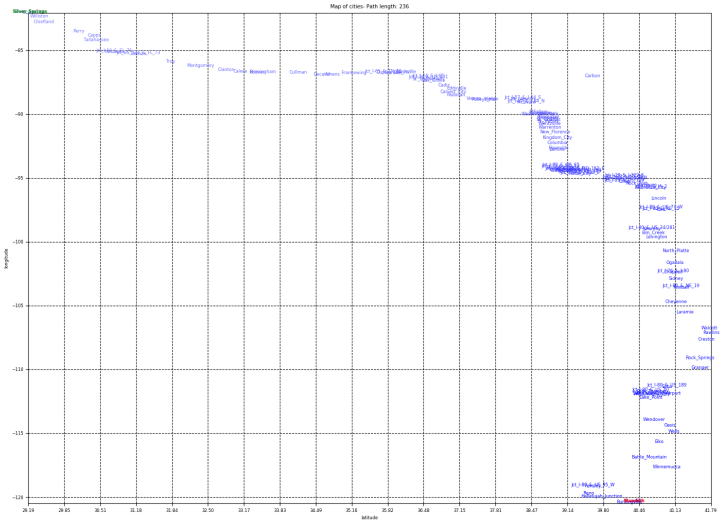
Problems we faced:
- Some of the gps in the city-gps file for some cities are missing. This means that we cannot calculate heuristic function accurately for the current city. To face this problem, we assumed that the heuristic for such a city is minimum value. This would guarantee that this city would not lose the chance of being removed from the fringe and we would not miss any possible candidate for the best route.
- Some of the highways do not have speed limit. To solve this problem we ignore those highway so they would not show in our best found paths as an answer.
Design decisions:
- To make the code shorter we used one function for couple of routing algorithm and one function for calculating g(s) of any given cost. This would make debugging and reading easier.
- Visualization of the path is possible by adding one extra word *yes* at the end of the input arguments for calling the *rout.py* file. This helps to find any possible pitfalls or bugs and understand the way that algorithm works.

Results:
After comparing, no difference in result was observed for the three heuristic functions in astar search.
As we can see in Figure 1, for finding the best path from *Standish,_California* to *Silver_Springs,_Florid*, astar search prefers longer distances with fewer cities for segments cost function while for distance cost function it chooses a more straight path which would result in a shorter distance. However for time cost function the path includes lots of cities which are close to each-other since astar tried to find highways with higher speed limits and shorter length.
Although the heuristic function is not admissible for time and segments but its result is comparable to the uniform search as we can see in table 1. As the table notes, astar has the same answer for distance cost function since its admissible. But for time and segments since it is not admissible, the result are different from uniform search algorithm.

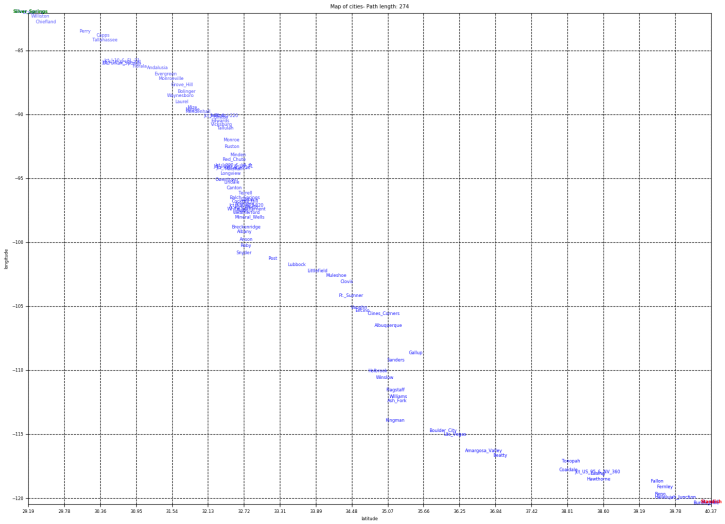
Table 1: astar vs uniform search

		Cost	
		astar	uniform
cost func	segments	71	68
	distance(mi)	2737	2737
	time(hr)	47.09	45.49

(a) astar - Time



(b) astar - Distance



(c) astar - Segments

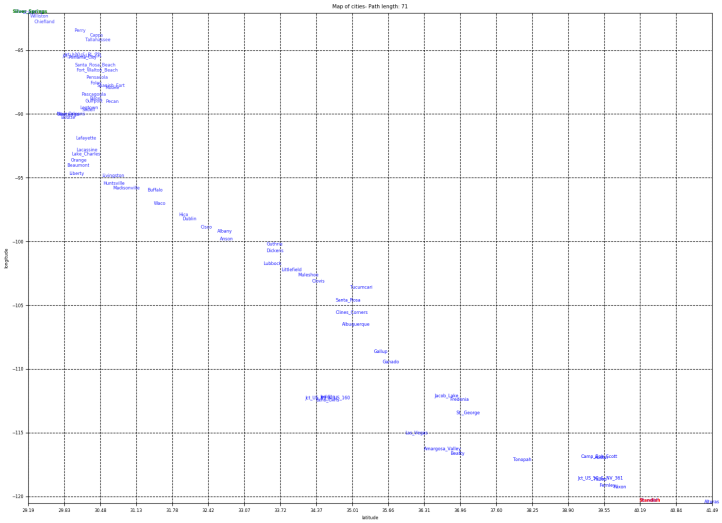


Figure 1: Comparison of astar search for different cost function(Section a: Time, Section b: Distance, Section c: Segments