Elements of Artificial Intelligence (CSCI - B551)
Assignment 1 - Problem 1

Submitted by: bchalasa-aralshi-dhsingh

1. Which search algorithm seems to work best for each routing options?

For the objective of finding the most optimal path, A* outperforms rest of the algorithms and gives the most optimal path. To measure performance and ensure consistency, we tested on multiple sets of start city and end city. Below is the observation table that illustrates the results we got corresponding to each algorithm for two test runs: one between near by cities (start city: Minturn,_Colorado; end city: Denver,_Colorado) and another between far away cities (start city: Los_Angeles,_California ; end city: Bangor,_Maine)

Case 1: Start City: Los_Angeles,_California End City: Bangor,_Maine

Cost Function	A*	Uniform Cost	BFS	DFS
Distance (miles)	3,254	4,247	5,611	36,210
Time (hours)	56.7401943755	66.6135432489	114.25758675	750.63532045
Segments (#edge)	72	72	72	1597
Longtour (miles)	41,893	6,871	5,611	36,210

Case 2: Start City: Minturn,_Colorado End City: Denver,_Colorado

Cost Function	A*	Uniform Cost	BFS	DFS
Distance (miles)	96	173	205	318
Time (hours)	1.47692307692	1.47692307692	4.50769230769	6.93675213675
Segments (#edge)	6	6	6	9
Longtour (miles)	206	206	205	318

Table 1: Results of Algorithms with different cost function

2. Which algorithm is fastest in terms of the amount of computation time required by your program, and by how much, according to your experiments?

For overall runtime, all algorithms are doing pretty good. A* takes longer time in comparison to rest all. This behavior can be attributed to the re-visiting of states since we are using an admissible heuristic with search algorithm# 2. To enhance the runtime performance of A*, we can multiply the heuristic with a certain factor and thus increasing the value of it which would make A* to visit less nodes in comparison and subsequently decreasing the overall runtime. While doing so we are compromising over the optimal behavior of A*. Therefore, it's a trade off between runtime and how optimal solution you are looking for? We performed multiple experiments and tested range of

multiplying factors and compared runtime and optimal result, below is the table that shows result for few of those iterations. Finally, we settled upon a multiplying factor of 1.2 if start city and end city are in different states and keeping it same if both are in same state; resulting in drastic reduction in runtime and maintaining the optimality at the same time. We recorded run time of all algorithms with different cost functions for above two case. From the observations below, we can assert that DFS runs faster in comparison to rest of algorithms if cities are separated by considerable distance and if they are close to each other, all algorithms take pretty much similar time to execute.

Start City: Los_Angeles,_California End City: Bangor,_Maine

Multiplying factor	Distance (miles)	Runtime (sec)
1.2	3,254	2.78
1.5	3,447	1.039
2	3,561	0.890
5	3,752	0.887

Table 2: Distance (in miles) and Runtime (sec) for different multiplying factor of A*

Case 1: Start City: Los_Angeles,_California End City: Bangor,_Maine

Cost Function	A*	Uniform Cost	BFS	DFS
Distance	2.78	1.87	1.67	1.13
Time	2.97	1.86	1.66	1.13
Segments	41.49	1.78	1.68	1.12
Longtour	5.15	7.65	1.63	1.14

Case 2: Start City: Minturn,_Colorado End City: Denver,_Colorado

Cost Function	A*	Uniform Cost	BFS	DFS
Distance	0.84	0.91	0.84	0.84
Time	0.85	0.85	0.86	0.87
Segments	0.85	0.87	0.84	0.85
Longtour	0.94	0.85	0.86	0.86

Table 3: Runtime (user time in seconds) of Algorithms

3. Which algorithm requires the least memory, and by how much, according to your experiments?

We tested the memory usage using ‘ /usr/bin/time -lp ’ utility and below are observed max resident set sizes (in MB) of above two cases for all algorithms with different cost functions. It seems like memory wise, BFS outperforms rest of the algorithms.

Case 1: Start City: Los_Angeles,_California End City: Bangor,_Maine

Cost Function	A*	Uniform Cost	BFS	DFS
Distance	28.93824	22.614016	17.440768	28.954624
Time	37.556224	22.728704	17.465344	28.893184
Segments	68.210688	20.905984	17.567744	29.069312
Longtour	54.996992	25.153536	17.59232	29.110272

Case 2: Start City: Minturn,_Colorado End City: Denver,_Colorado

Cost Function	A*	Uniform Cost	BFS	DFS
Distance	17.174528	17.117184	17.047552	17.317888
Time	17.170432	17.043456	17.092608	17.104896
Segments	17.28512	17.297408	17.063936	17.338368
Longtour	17.760256	17.432576	17.035264	17.211392

Table 4: Memory Usage (max resident set size in MB) of Algorithms

4. Which heuristic function(s) did you use, how good is it, and how might you make it/ them better?

For a heuristic, in case of distance cost function, we used Haversine distance. For each city, the actual traveling distance would always be greater than the haversine distance, so it meets the criteria of an admissible heuristic. But there were few cities and all junction for which we could not calculate the haversine distance since we didn’t have longitudes and latitudes for them. For those junctions, we took the nearby connected city and calculated heuristic of that city and then subtracted the actual distance from junction to the city. There were cases where junctions were connected to only junctions, for those cases we used 0 as a heuristic. We have multiplied our heuristic with a factor of 1.2 to make A* more efficient in terms of execution time.

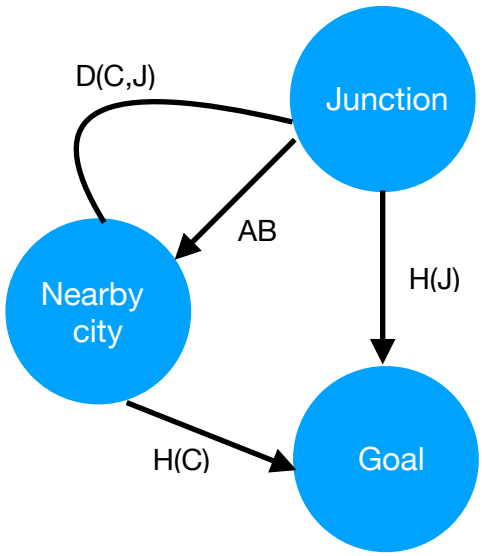

Figure 1: Heuristic Calculation of Junctions

In the above the diagram, consider the triangle formed by three straight lines:

H(C) - AB will always be smaller than the H(J) [Difference of two sides is always smaller than the third side]
=> H(C) - AB < H(J)

Now, we are subtracting the original distance (D(C.J)) which will always be greater than AB. Therefore -
=> H(C) - D(C,J) < H(J)

Hence, our heuristic would always be less than the original heuristic and if this becomes much smaller, A* will open many nodes, eventually becoming slower. As it is already very less than the original heuristic, we can multiply it by a small factor maintaining the admissibility. To make it better than the existing one, we could have done one more thing; for Junctions, which are connected to only junctions, we are using 0 heuristic. Instead, we could have used the distance of a nearest connected node because it will always be lesser than the distance to goal.

For time as cost function, we used the same distance heuristic and divided by the median speed. For segments, we used 1 as a heuristic. Since, we are adding same constant to each node, it will run similar to uniform cost and give same results. For longtour, we used the same distance heuristic.