Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS3340

Project 1 Report

Search for Optimal Service

Prepared by :

Tareq Shannak - 1181404

Abd Al-Rahman Mansour - 1182955

Instructor : Dr. Adnan Yahya

Section : 1

Date : 13/4/2021

Program Implementation

We used Java to implement our program using Eclipse, also we used Scene builder to make our fxml views. First, we created two classes: City and Node, Node has two attributes which are id and drive route, this class will be used to define streets between cities. City class has four attributes: id, cost, evaluation Function and the city next to, these attributes define a single city and has a pointer to the city which is next to. In main, we lunched the Home Page view and read the input file “AerialDistancesAndRoutes.txt” to know the aerial and practical distances between twenty cities which they are sorted by alphabetical order as shown on Table 1. After reading the input files, we put the aerial and practical distances in two array lists.

We have two java files: HomePage and ResultPage, each of them controls an fxml file to interface our work, home page make user selects which parameters he need, start city, goal cities and algorithm type. Result Page shows the path that founds a goal, the order of city expansion list, heuristic chart and some info about the selected algorithm.

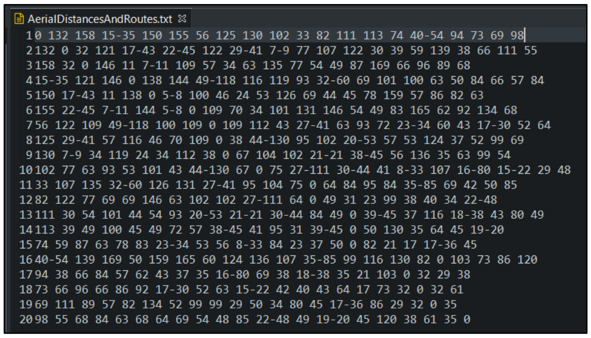
0	Aka
1	Bethlehem
2	Dura
3	Haifa
4	Halhoul
5	Hebron
6	Jenin
7	Jericho
8	Jerusalem
9	Nablus
10	Nazareth
11	Qalqilya
12	Ramallah
13	Ramleh
14	Sabastia
15	Safad
16	Salfit
17	Tubas
18	Tulkarm
19	Yafa

Table 1 - Cities and Their IDs

The java file which is called SearchAlgorithm.java has our algorithms to search for service. Every search method takes the start city’s id and an array list of the goals that we need to search. The algorithms that we required to implement: Greedy Search, Iterative Deepening and optimal 1 for all goals, but we also implemented extra four algorithms: Breadth First Search, Depth First Search, Uniform Cost Search and A* Searches. BFS uses a simple queue (FIFO) in fringe to check the cities if it is a goal or not unlike DFS which uses a stack (LIFO) so it searches in depth not in levels. IDS is a component search of DFS and BFS, we used a recursion method to implement this search which needs to search in level of the graph after search the depth to reach that’s level. Greedy, UCS and A* Searches have the same idea in implementation which they use a priority queue that sorts the elements in the queue according to the city’s evaluation function in ascending order. The difference between those three searches is how to calculate the evaluation function for each city. We assumed the evaluation function is F(n) = G(n) + H(n), in greedy search: G(n)=0 and in UCS: H(n)=0. The last algorithm is the optimal search for all goals which deals with the distance as the main parameter to be optimized, it uses A* algorithm to search for the nearest goal, after that it searches for the second nearest goal starting from the previous goal found and so on until we found all goals.

How the Program Runs

First, we need to write the aerial and practical distances between cities in the input text file as shown on Figure 1, we put a default data which we obtained from google maps.



AerialDistancesAndRoutes.txt

```
0 132 158 15-35 150 155 56 125 130 102 33 82 111 113 74 40-54 94 73 69 98
132 0 32 121 17-43 22-45 122 29-41 7-9 77 107 122 30 39 59 139 38 66 111 55
158 32 0 146 11 7-11 109 57 34 63 135 77 54 49 87 169 66 96 89 68
15-35 121 146 0 138 144 49-118 116 119 93 32-60 69 101 100 63 50 84 66 57 84
150 17-43 11 138 0 5-8 100 46 24 53 126 69 44 45 78 159 57 86 82 63
155 22-45 7-11 144 5-8 0 109 70 34 101 131 146 54 49 83 165 62 92 134 68
56 122 109 49-118 100 109 0 109 112 43 27-41 63 93 72 23-34 60 43 17-30 52 64
125 29-41 57 116 46 70 109 0 38 44-130 95 102 20-53 57 53 124 37 52 99 69
130 7-9 34 119 24 34 112 38 0 67 104 102 21-21 38-45 56 136 35 63 99 54
102 77 63 93 53 101 43 44-130 67 0 75 27-111 30-44 41 8-33 107 16-80 15-22 29 48
33 107 135 32-60 126 131 27-41 95 104 75 0 64 84 95 84 35-85 69 42 50 85
82 122 77 69 69 146 63 102 102 27-111 64 0 49 31 23 99 38 40 34 22-48
111 30 54 101 44 54 93 20-53 21-21 30-44 84 49 0 39-45 37 116 18-38 43 80 49
113 39 49 100 45 49 72 57 38-45 41 95 31 39-45 0 50 130 35 64 45 19-20
74 59 87 63 78 83 23-34 53 56 8-33 84 23 37 50 0 82 21 17 17-36 45
40-54 139 169 50 159 165 60 124 136 107 35-85 99 116 130 82 0 103 73 86 120
94 38 66 84 57 62 43 37 35 16-80 69 38 18-38 35 21 103 0 32 29 38
73 66 96 66 86 92 17-30 52 63 15-22 42 40 43 64 17 73 32 0 32 61
69 111 89 57 82 134 52 99 99 29 50 34 80 45 17-36 86 29 32 0 35
98 55 68 84 63 68 64 69 54 48 85 22-48 49 19-20 45 120 38 61 35 0
```

Figure 1 – Data

When we press run button, we will see the home page, when we select an algorithm type from the combo box, the progress bar shows us a progress as shown on Figure 2.

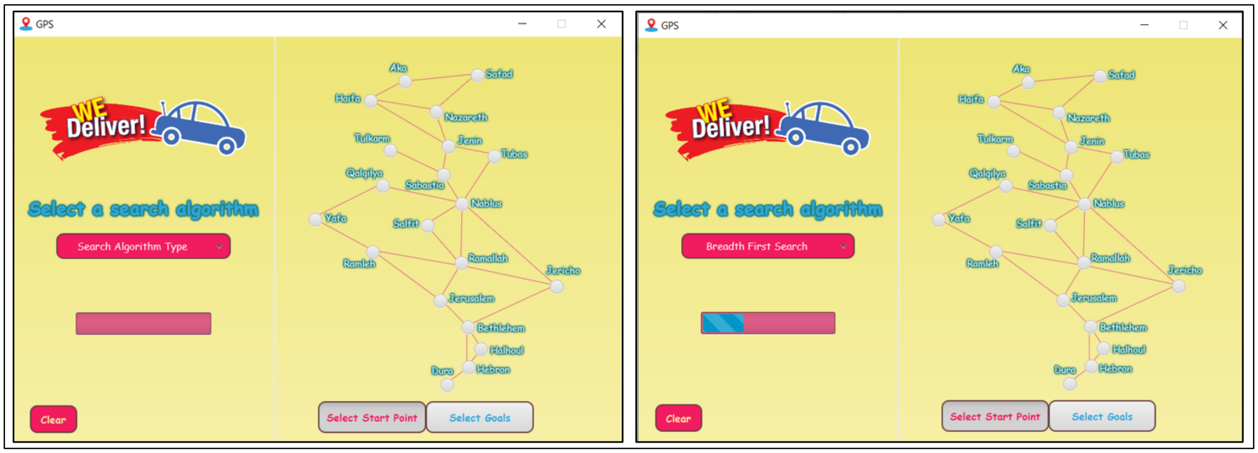


Figure 2

After that we select a start city and save it, the progress will increase in the bar as shown on Figure 3.

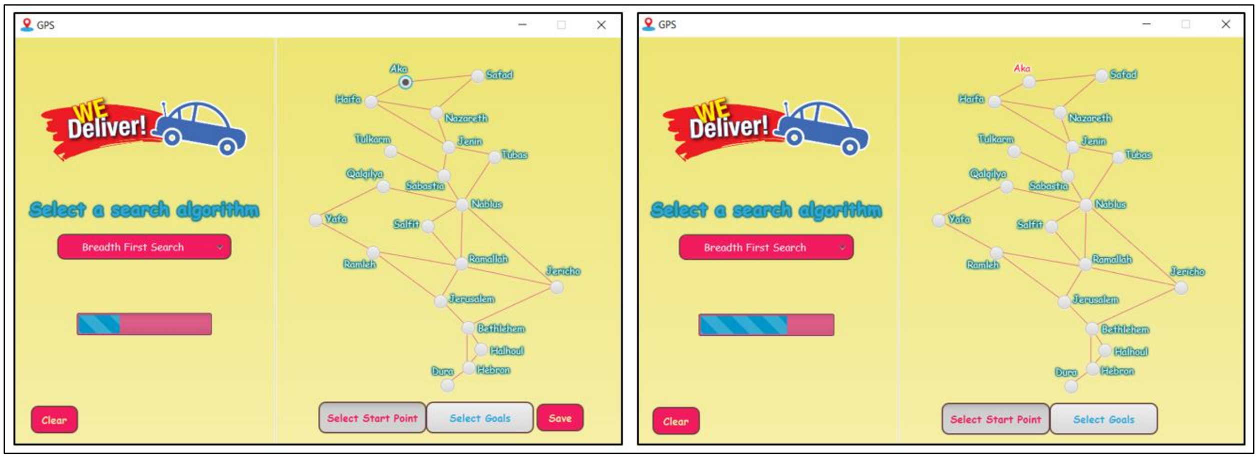


Figure 3

Figure 4 shows us the progress bar when is full, it will disappear a search button after we select the goals and save them.

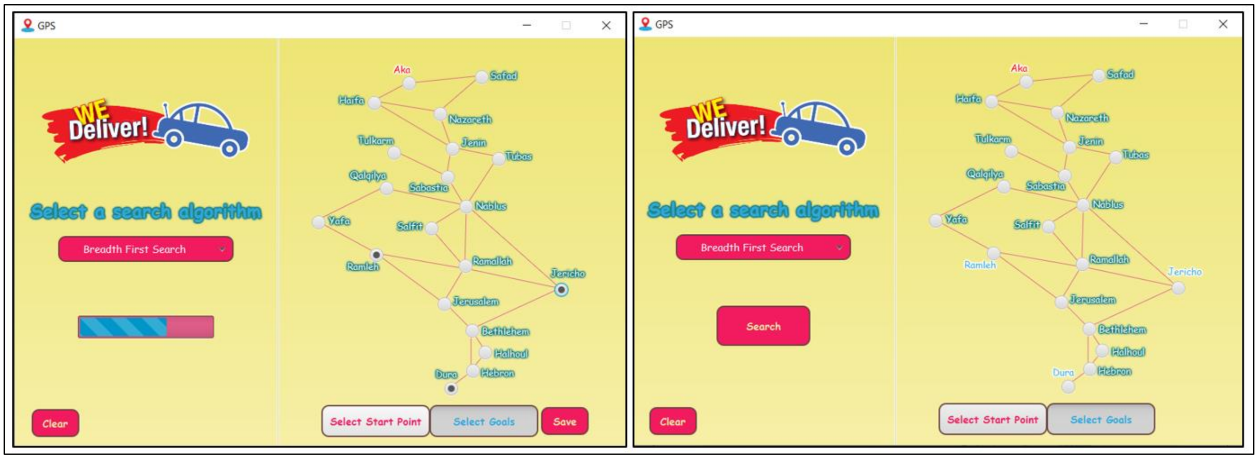
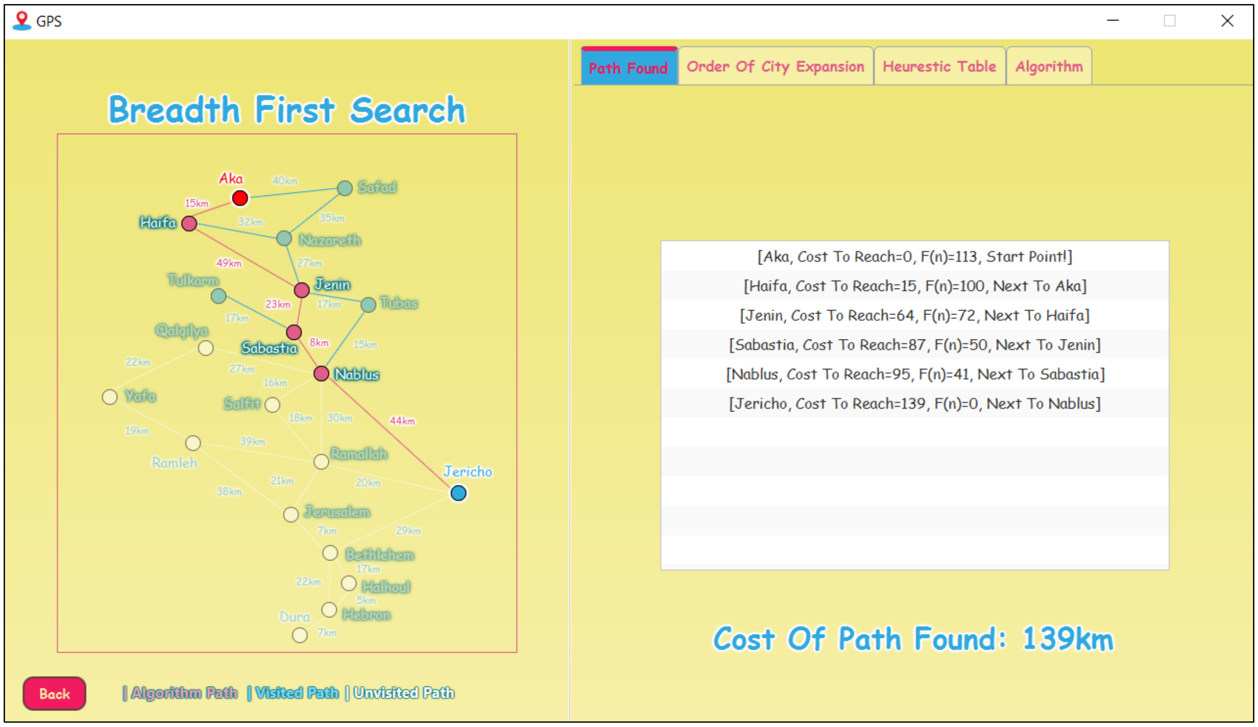


Figure 4

Figure 5 shows the map and paths in three colors, our true path is in red color, visited nodes and paths (but not in the true path of the algorithm) are in blue color and the white nodes and routes are neither visited nor found in our true path.

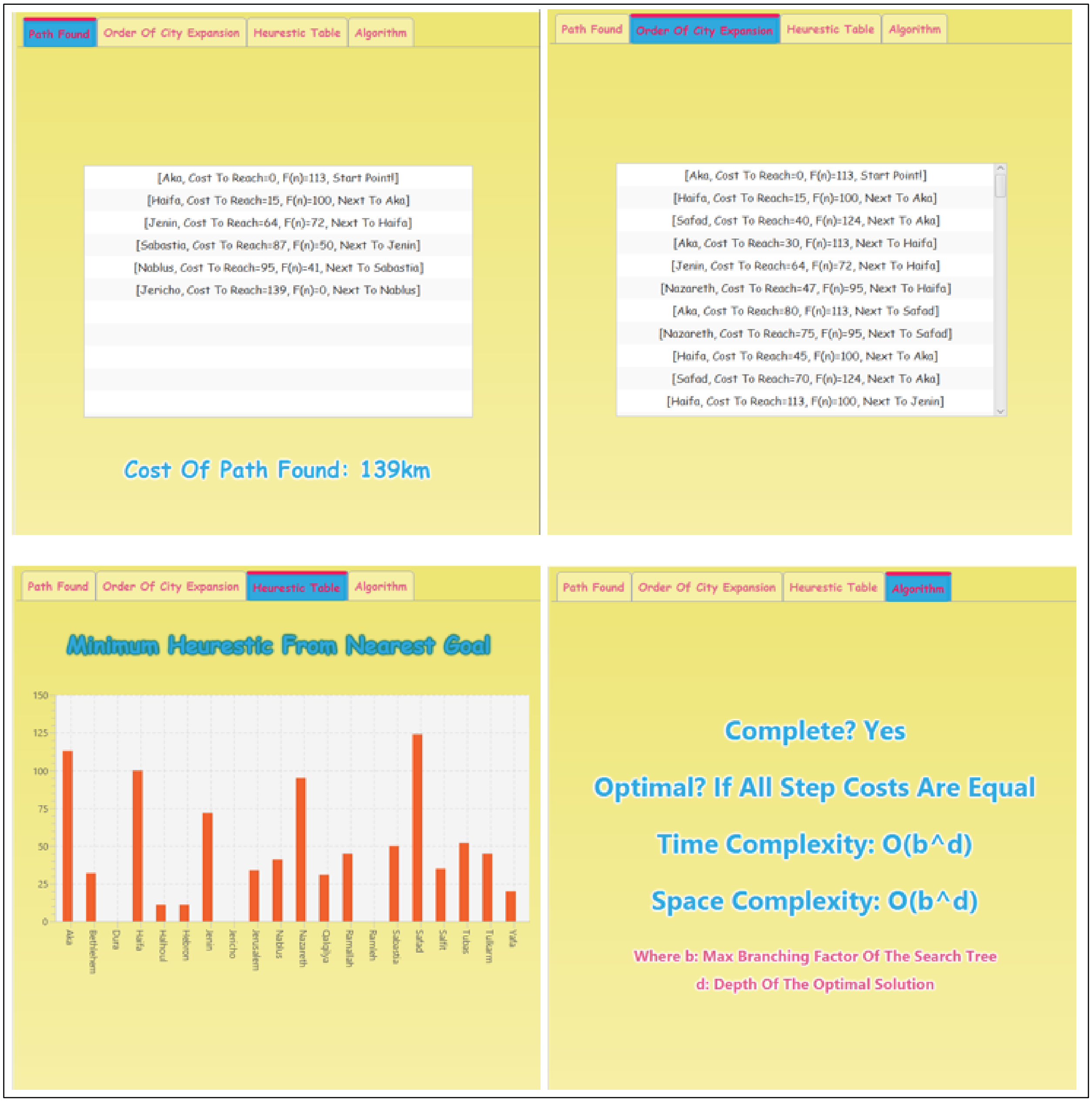


[Aka, Cost To Reach=0, F(n)=113, Start Point!]
[Haifa, Cost To Reach=15, F(n)=100, Next To Aka]
[Jenin, Cost To Reach=64, F(n)=72, Next To Haifa]
[Sabastia, Cost To Reach=87, F(n)=50, Next To Jenin]
[Nablus, Cost To Reach=95, F(n)=41, Next To Sabastia]
[Jericho, Cost To Reach=139, F(n)=0, Next To Nablus]

Cost Of Path Found: 139km

Figure 5

Figures 6 shows the tab pane in the right of the result page that contain tabes show the order of true cities path, cost of path found, order of cities expansion, heuristic table and info about the chosen algorithm.



[Aka, Cost To Reach=0, F(n)=113, Start Point!]
[Haifa, Cost To Reach=15, F(n)=100, Next To Aka]
[Jenin, Cost To Reach=64, F(n)=72, Next To Haifa]
[Sabastia, Cost To Reach=87, F(n)=50, Next To Jenin]
[Nablus, Cost To Reach=95, F(n)=41, Next To Sabastia]
[Jericho, Cost To Reach=139, F(n)=0, Next To Nablus]

Cost Of Path Found: 139km

[Aka, Cost To Reach=0, F(n)=113, Start Point!]
[Haifa, Cost To Reach=15, F(n)=100, Next To Aka]
[Safad, Cost To Reach=40, F(n)=124, Next To Aka]
[Aka, Cost To Reach=30, F(n)=113, Next To Haifa]
[Jenin, Cost To Reach=64, F(n)=72, Next To Haifa]
[Nazareth, Cost To Reach=47, F(n)=95, Next To Haifa]
[Aka, Cost To Reach=80, F(n)=113, Next To Safad]
[Nazareth, Cost To Reach=75, F(n)=95, Next To Safad]
[Haifa, Cost To Reach=45, F(n)=100, Next To Aka]
[Safad, Cost To Reach=70, F(n)=124, Next To Aka]
[Haifa, Cost To Reach=113, F(n)=100, Next To Jenin]

Complete? Yes

Optimal? If All Step Costs Are Equal

Time Complexity: O(b^d)

Space Complexity: O(b^d)

Where b: Max Branching Factor Of The Search Tree
d: Depth Of The Optimal Solution

Figure 6