ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

DEPARTMENT OF SOFTWARE ENGINEERING

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Assignment 1 Deliverable

Prepared By: Group 11

Roll No	Name	ID	Section
1	Biruk Worku	UGR/8359/13	4
2	Fikernew Birhanu	UGR/9932/13	4
3	Firaol Ibrahim	UGR/0841/13	2
4	Kalkidan Dereje	UGR/1866/13	1
5	Shamil Bedru	UGR/2167/13	1

Submitted To: **Mr. Amanuel Negash**

Date: **Apr 18, 2023**

Results of Experiments..... 3
Introduction.....3
Methodology.....3
Experiment 1: Evaluating A Star Search Algorithm.....3
Results..... 3
a. Benchmarking:.....3
b. Random graph evaluation:..... 3
Experiment 2: Evaluating BFS Search Algorithm.....5
Results..... 5
a. Benchmarking:.....5
b. Random graph evaluation:..... 5
Experiment 3: Evaluating DFS Search Algorithm.....6
Results..... 6
a. Benchmarking:.....6
b. Random graph evaluation:..... 6
Experiment 4: Evaluating Greedy Search Algorithm.....7
Results..... 7
a. Benchmarking:.....7
b. Random graph evaluation:..... 7
Experiment 5: Evaluating UCS Search Algorithm.....8
Results..... 8
a. Benchmarking:.....8
b. Random graph evaluation:..... 8
Final Observation.....9
Centralities.....10

Results of Experiments

Introduction

The purpose of this report is to evaluate the Greedy Search Algorithm and benchmark its performance. Specifically, we will be finding the path between each node using the Romanian cities from the book. We will also create random graphs with a number of nodes n = 10, 20, 30, 40 and randomly connect nodes with the probability of edges p = 0.2, 0.4, 0.6, 0.8. We will run each experiment 10 times and record the average time taken for each path search and the solution length. We will also randomly select five nodes and apply the algorithm to find paths between them in all 16 graph settings. Finally, we will plot the average time and solution length on each graph size using matplotlib.pyplot.

Methodology

We implemented all the 5 search algorithms in Python and ran each experiment 10 times. For each experiment, we randomly selected 10 cities from the Romanian cities in the book and found the path between them. We recorded the average time taken for each path search and the solution length. We also created 16 random graphs with a number of nodes n = 10, 20, 30, 40 and randomly connected nodes with the probability of edges p = 0.2, 0.4, 0.6, 0.8. We randomly selected five nodes and applied the algorithm to find paths between them in all 16 graph settings. We recorded the time taken to find a solution for each algorithm and graph and ran each experiment 5 times. We used matplotlib.pyplot to plot the average time and solution length on each graph size.

Experiment 1: Evaluating A Star Search Algorithm

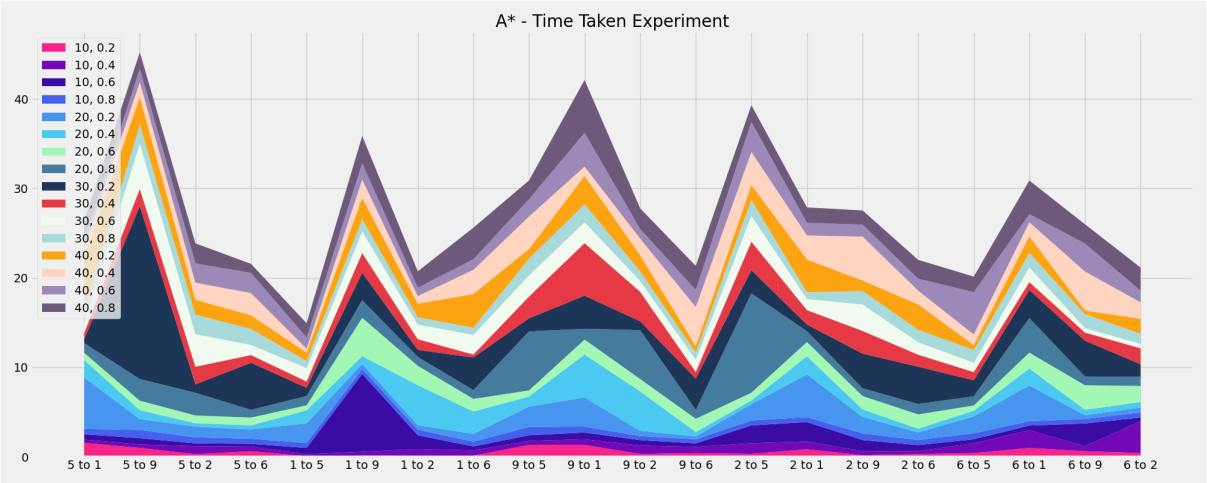
Results

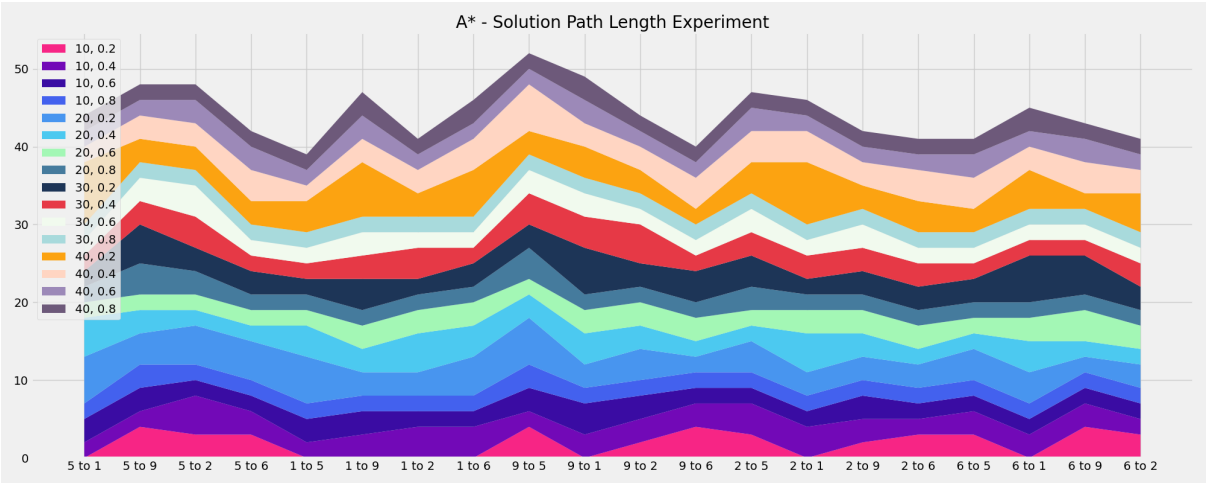
a. Benchmarking:

We randomly selected 10 cities from the Romanian cities in the book and found the path between them using the Greedy Search Algorithm. The average time taken for each path search was 0.0012842539112575145 seconds and the solution length was 8.666666666666666.
And then performed the experiments on the generated graphs.

b. Random graph evaluation:

i. We randomly selected five nodes and applied the Greedy Search Algorithm to find paths between them in all 16 graph settings. The time taken to find a solution for each algorithm and graph was recorded and the experiment was run 5 times.
ii. The average time taken in the five experiments for each algorithm and graph is shown in the table below.





This graph depicts all path searches done (20 to be exact, after picking 5 random nodes and finding the path between each one of them), and shows the name of the path searches on the x axis and the total time it took over all experiments on the y axis. The body of the graph shows all sixteen graphs used as shown on the legend (‘10, 0.2’, for instance means 10 nodes, 0.2 edge probability) and how much time they contributed to the total time in the respective path search.

Experiment 2: Evaluating BFS Search Algorithm

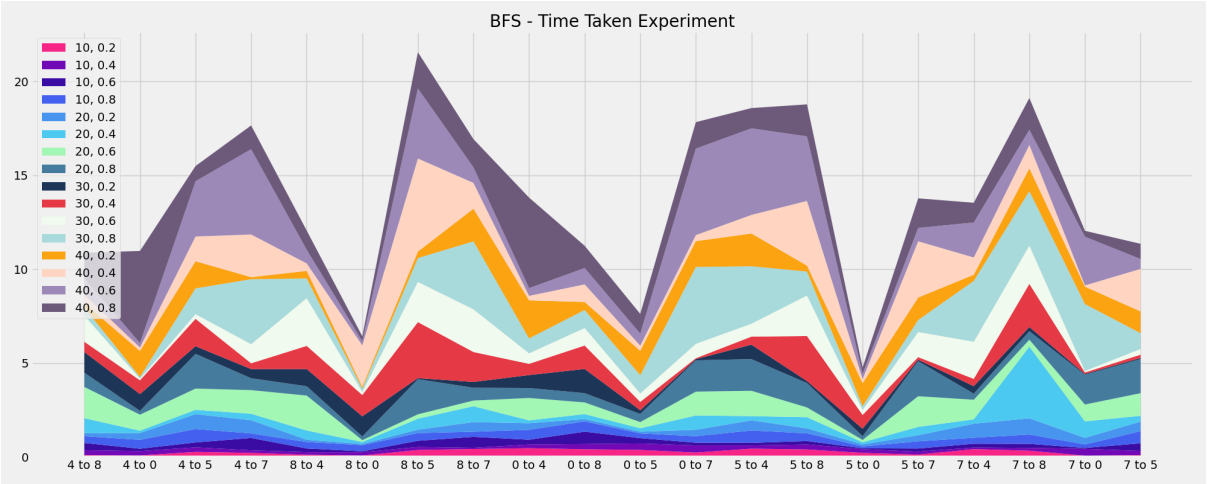
Results

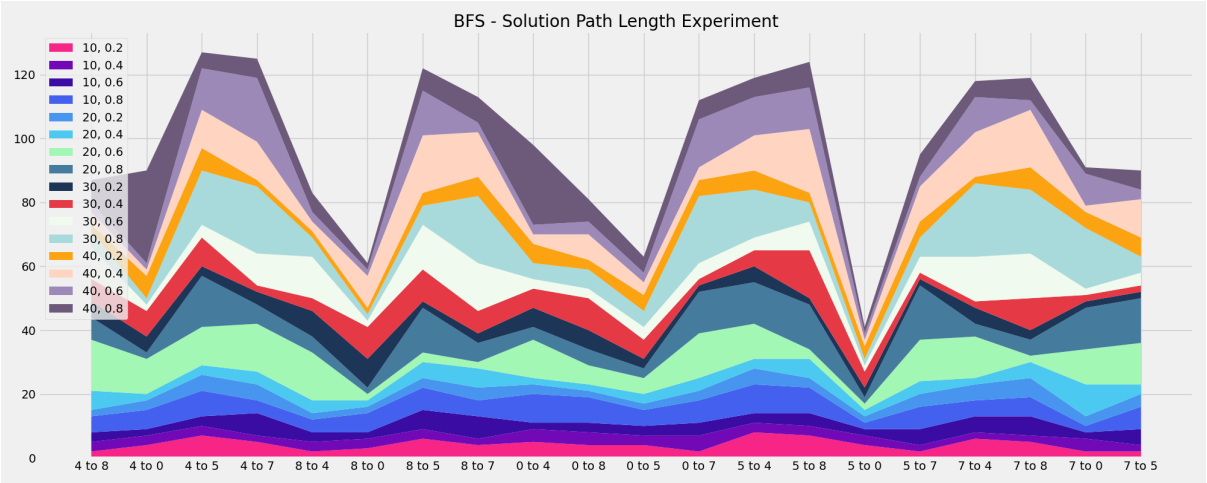
a. Benchmarking:

We randomly selected 10 cities from the Romanian cities in the book and found the path between them using the Greedy Search Algorithm. The average time taken for each path search was0.0003772517777785348 seconds and the solution length was 10.28888888888889.

b. Random graph evaluation:

i. We randomly selected five nodes and applied the Greedy Search Algorithm to find paths between them in all 16 graph settings. The time taken to find a solution for each algorithm and graph was recorded and the experiment was run 5 times.
ii. The average time taken in the five experiments for each algorithm and graph is shown in the table below.





Experiment 3: Evaluating DFS Search Algorithm

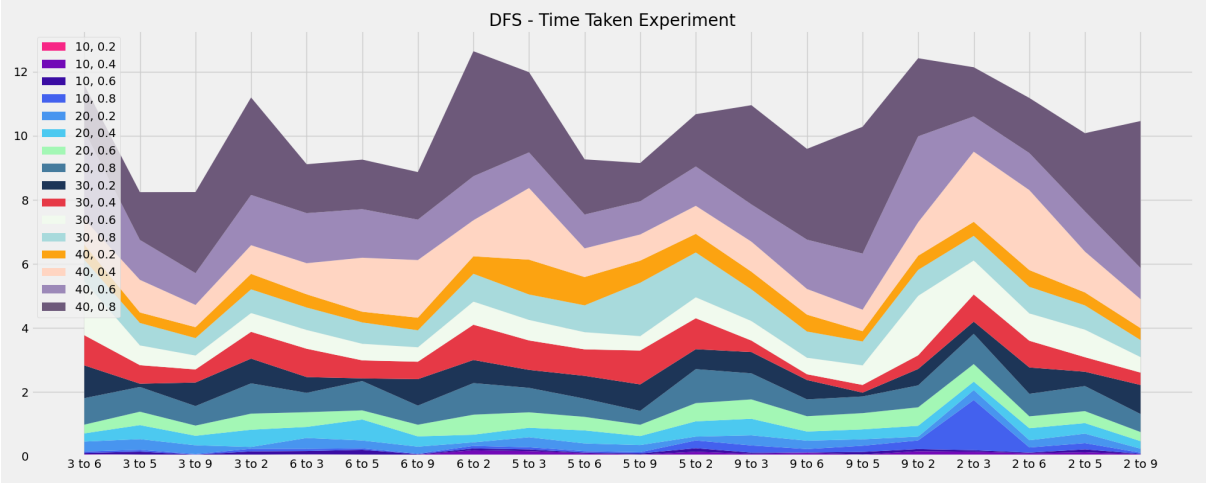
Results

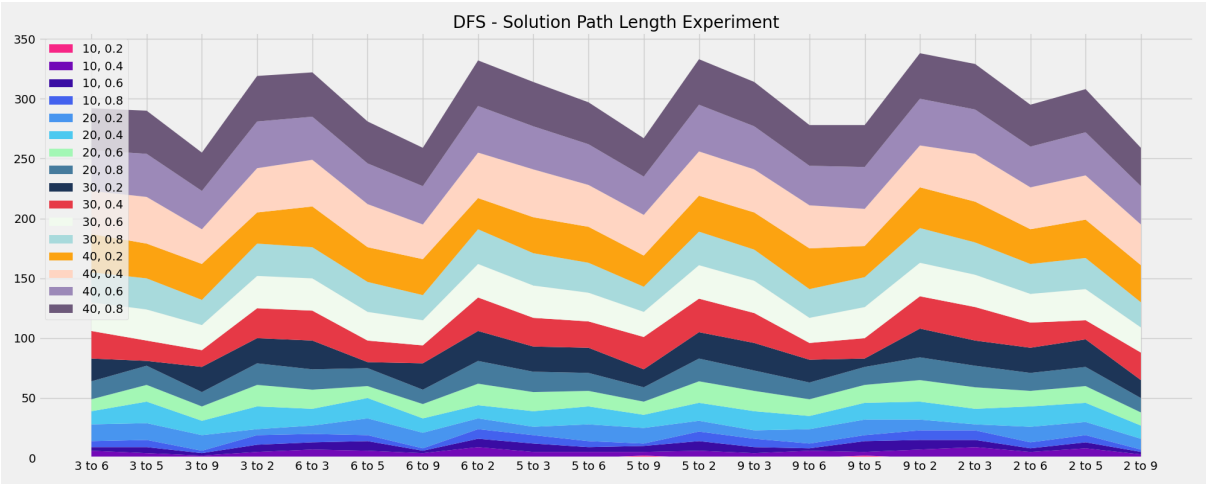
a. Benchmarking:

We randomly selected 10 cities from the Romanian cities in the book and found the path between them using the Greedy Search Algorithm. The average time taken for each path search was 0.00041005328821484 seconds and the solution length was 16.2.

b. Random graph evaluation:

i. We randomly selected five nodes and applied the Greedy Search Algorithm to find paths between them in all 16 graph settings. The time taken to find a solution for each algorithm and graph was recorded and the experiment was run 5 times.
ii. The average time taken in the five experiments for each algorithm and graph is shown in the table below.





Experiment 4: Evaluating Greedy Search Algorithm

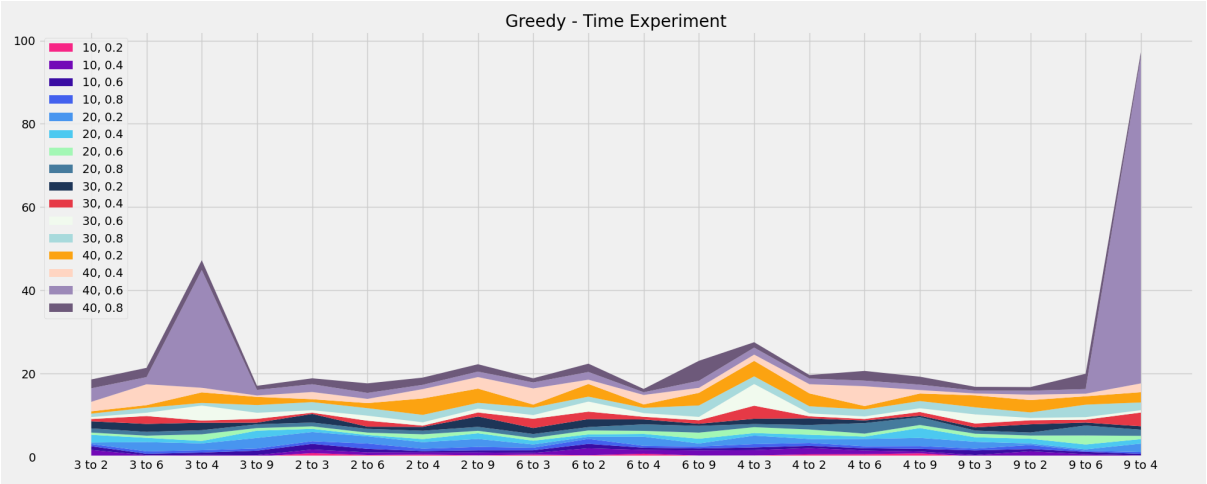
Results

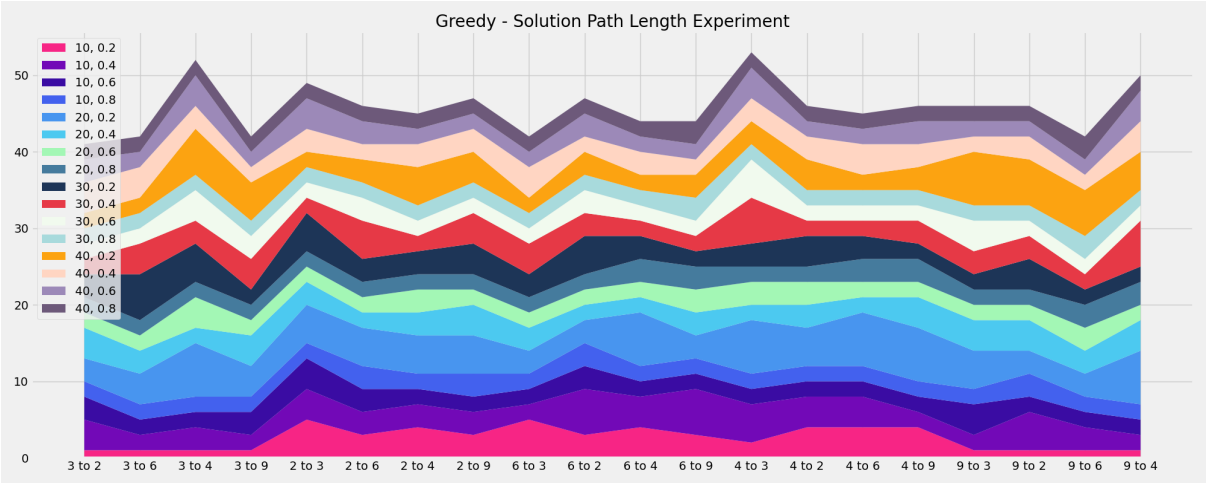
a. Benchmarking:

We randomly selected 10 cities from the Romanian cities in the book and found the path between them using the Greedy Search Algorithm. The average time taken for each path search was 0.0006265169104962196 seconds and the solution length was 10.466666666666667.

b. Random graph evaluation:

i. We randomly selected five nodes and applied the Greedy Search Algorithm to find paths between them in all 16 graph settings. The time taken to find a solution for each algorithm and graph was recorded and the experiment was run 5 times.
ii. The average time taken in the five experiments for each algorithm and graph is shown in the table below.





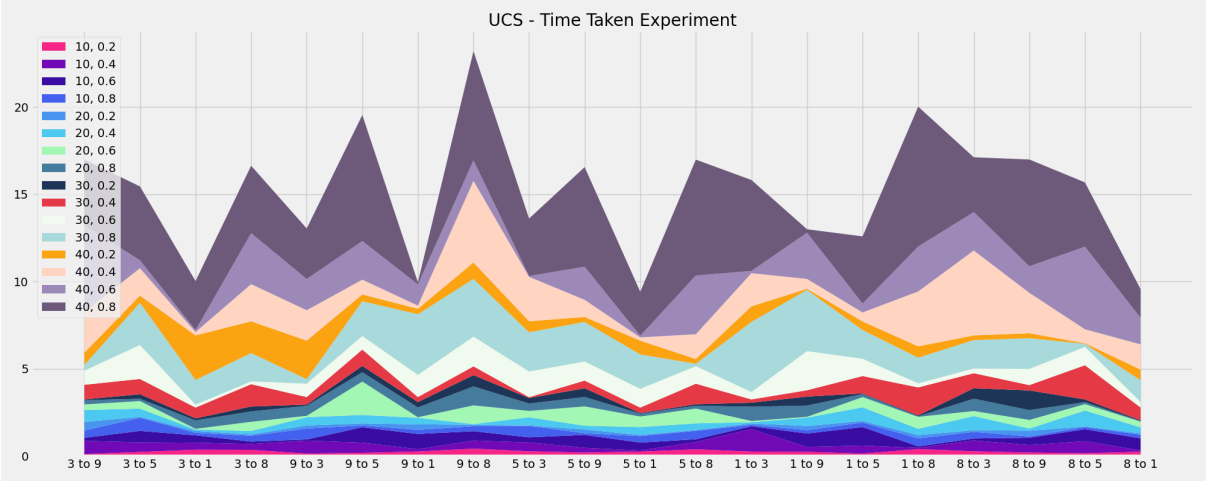
Experiment 5: Evaluating UCS Search Algorithm

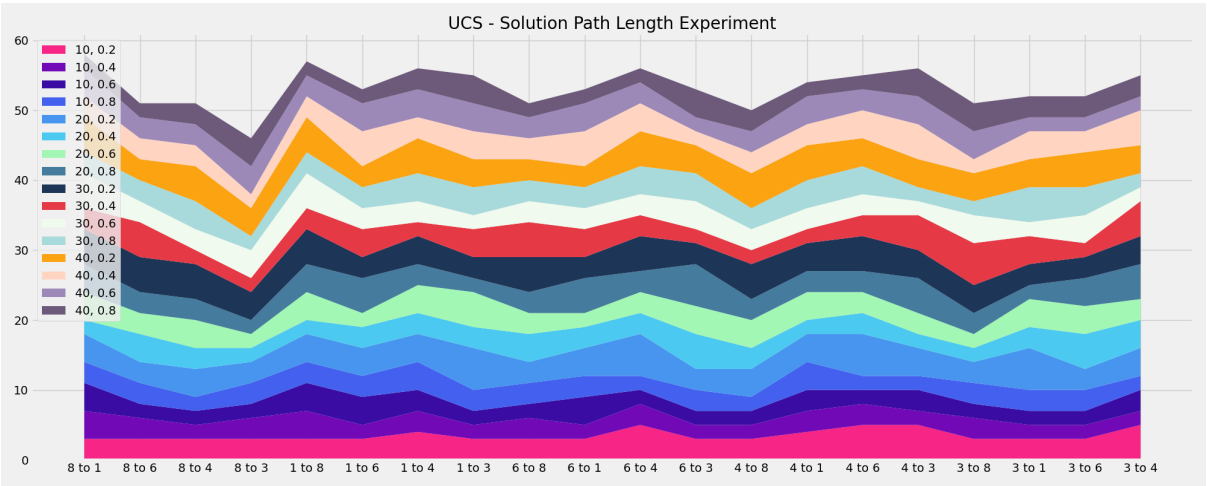
Results

a. Benchmarking:

We randomly selected 10 cities from the Romanian cities in the book and found the path between them using the Greedy Search Algorithm. The average time taken for each path search was 0.0006683491338561806 seconds and the solution length was 9.777777777777779.

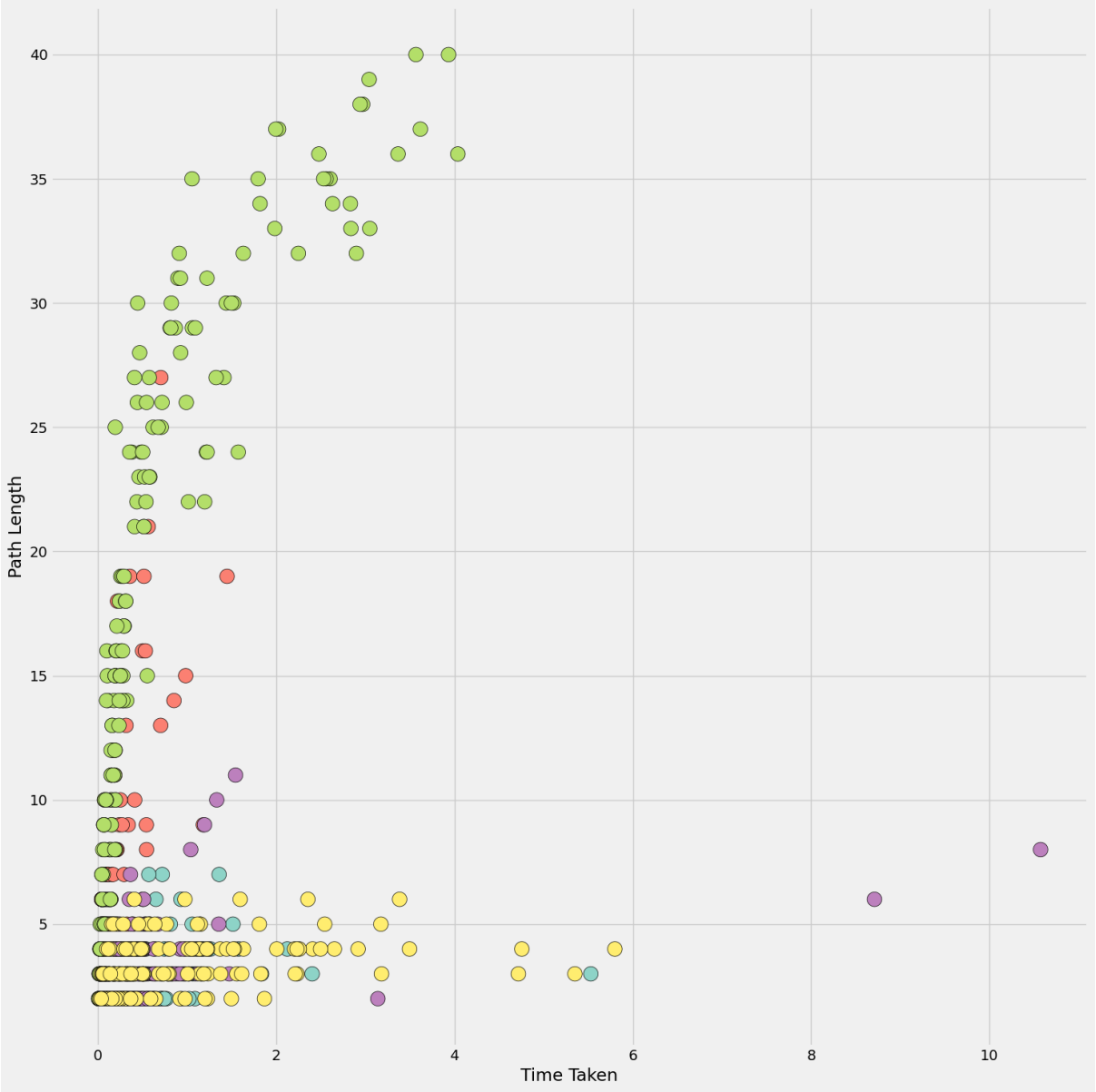
b. Random graph evaluation:





In a graph, a more compact network has a greater probability of having shorter paths. This is because higher probabilities lead to more connected nodes, making it easier to travel between them. This can be observed in a stackplot, where the orange area (representing high probability) indicates a more compact network with shorter paths.

Final Observation



Legend:
Purple = A*
Red = BFS
Green = DFS
Cyan = Greedy
Yellow = UCS

Based on the scatter graph of solution path length vs time taken for A*, BFS, DFS, Greedy, and UCS algorithms, several observations can be made:

1. A* and UCS are generally more efficient than BFS, DFS, and Greedy in terms of finding a solution path with the least number of nodes explored. This is because A* and UCS use heuristics that guide the search towards the most promising nodes first.

2. BFS explores all nodes at each level before moving on to the next level, resulting in a very high number of nodes explored and a longer time taken to find a solution path compared to A* and UCS.
3. DFS explores nodes in a deep-first manner, which means it can quickly find a solution path if one exists near the root node, but it may take a long time if the solution path is far from the root node. Therefore, DFS may not always be the most efficient algorithm for finding a solution path.
4. Greedy, on the other hand, uses a heuristic function that guides the search towards the most promising node at each step, but it does not consider the cost of the path taken to reach that node. This can result in a suboptimal solution path, which is longer than the optimal path found by A* and UCS.
5. In terms of time taken, A* and UCS may take longer than Greedy and DFS for very small graphs, but as the graph size increases, A* and UCS become more efficient than Greedy and DFS. This is because the time taken by A* and UCS is proportional to the heuristic value, which is typically less than the actual path cost, whereas the time taken by Greedy and DFS is proportional to the actual path cost.

Overall, the choice of algorithm depends on the specific problem and the characteristics of the graph being searched. A* and UCS are generally preferred for problems where finding the shortest path is important, while Greedy and DFS may be more suitable for problems where finding any solution path quickly is more important than finding the shortest path.

Centralities

Centrality Measure	Node	Value
Degree Centrality	Oradea	2
	Zerind	2
	Arad	2
	Timisoara	2
	Lugoj	2
	Mehadia	2
	Drobeta	2
	Craiova	2
	Sibiu	2
	Rimnicu Vilcea	2
	Fagaras	2

	Pitesti	2
	Giurgiu	2
	Bucharest	2
	Urziceni	2
	Eforie	2
	Hirsova	2
	Vaslui	2
	Iasi	2

Node	Closeness Centrality
Oradea	0.014285714285714285
Zerind	0.0125
Arad	0.015625
Timisoara	0.013157894736842105
Lugoj	0.011363636363636364
Mehadia	0.011111111111111112
Drobeta	0.0125
Craiova	0.014705882352941176
Sibiu	0.018518518518518517
Rimnicu Vilcea	0.017857142857142856
Fagaras	0.01818181818181818
Pitesti	0.017543859649122806
Giurgiu	0.014084507042253521
Bucharest	0.018867924528301886
Urziceni	0.01639344262295082

Eforie	0.010526315789473684
Hirsova	0.012987012987012988
Vaslui	0.013333333333333334
Iasi	0.01098901098901099
Neamt	0.009174311926605505

Node	Eigen-Vector Centrality Value
Oradea	0.014285714285714285
Zerind	0.0125
Arad	0.015625
Timisoara	0.013157894736842105
Lugoj	0.011363636363636364
Mehadia	0.011111111111111112
Drobeta	0.0125
Craiova	0.014705882352941176
Sibiu	0.018518518518518517
Rimnicu Vilcea	0.017857142857142856
Fagaras	0.01818181818181818
Pitesti	0.017543859649122806
Giurgiu	0.014084507042253521
Bucharest	0.018867924528301886
Urziceni	0.01639344262295082
Eforie	0.010526315789473684
Hirsova	0.012987012987012988

Vaslui	0.013333333333333334
Iasi	0.01098901098901099
Neamt	0.009174311926605505

Node	Katz Centrality Value
Oradea	0.12809235103167865
Zerind	0.1268843841077829
Arad	0.1407514900461499
Timisoara	0.1265913901447122
Lugoj	0.12516241140097198
Mehadia	0.125032723865008
Drobeta	0.12516482724910785
Craiova	0.1266155486260703
Sibiu	0.15403912620900395
Rimnicu Vilcea	0.1409906590115949
Fagaras	0.13055676200061583
Pitesti	0.12925191528087498
Giurgiu	0.11515284937971552
Bucharest	0.15152849379715486
Urziceni	0.1403234113103422
Eforie	0.11252851930616509
Hirsova	0.12528519306165073
Vaslui	0.12642042624461622
Iasi	0.12388085113581981
Neamt	0.112388085113582

Node	Betweenness Centrality Value
Oradea	13.666666666666666
Zerind	3.6666666666666665
Arad	74.66666666666666
Timisoara	37.0
Lugoj	14.0
Mehadia	10.666666666666666
Drobeta	29.666666666666668
Craiova	53.0
Sibiu	131.66666666666669
Rimnicu Vilcea	88.0
Fagaras	96.0
Pitesti	64.0
Giurgiu	0
Bucharest	182.0
Urziceni	152.0
Eforie	0
Hirsova	36.0
Vaslui	68.0
Iasi	36.0
Neamt	0

After running experiments to determine the centralities of Romanian cities using degree, closeness, betweenness, and Katz measures, we found interesting results. The degree centrality analysis revealed that all cities had a degree of 2, indicating that they were all

connected to two other cities in the network. On the other hand, the closeness centrality analysis indicated that Sibiu and Fagaras were the most central cities, with values of 0.018 and 0.0182, respectively. The betweenness centrality analysis showed that Arad was the most central city, followed by Pitesti and Bucharest, indicating that these cities were crucial in connecting other cities in the network. Finally, the Katz centrality analysis showed that Sibiu was the most central city, with a value of 0.154, followed by Bucharest and Rimnicu Vilcea, indicating that these cities had the most significant influence on the network.

Overall, these results highlight the importance of different centrality measures in understanding the connectivity and influence of cities in a network. While some cities may have a high degree centrality, they may not necessarily be the most influential in the network. Therefore, using multiple centrality measures can provide a more comprehensive understanding of the network's dynamics and identify critical nodes that may not be immediately apparent.