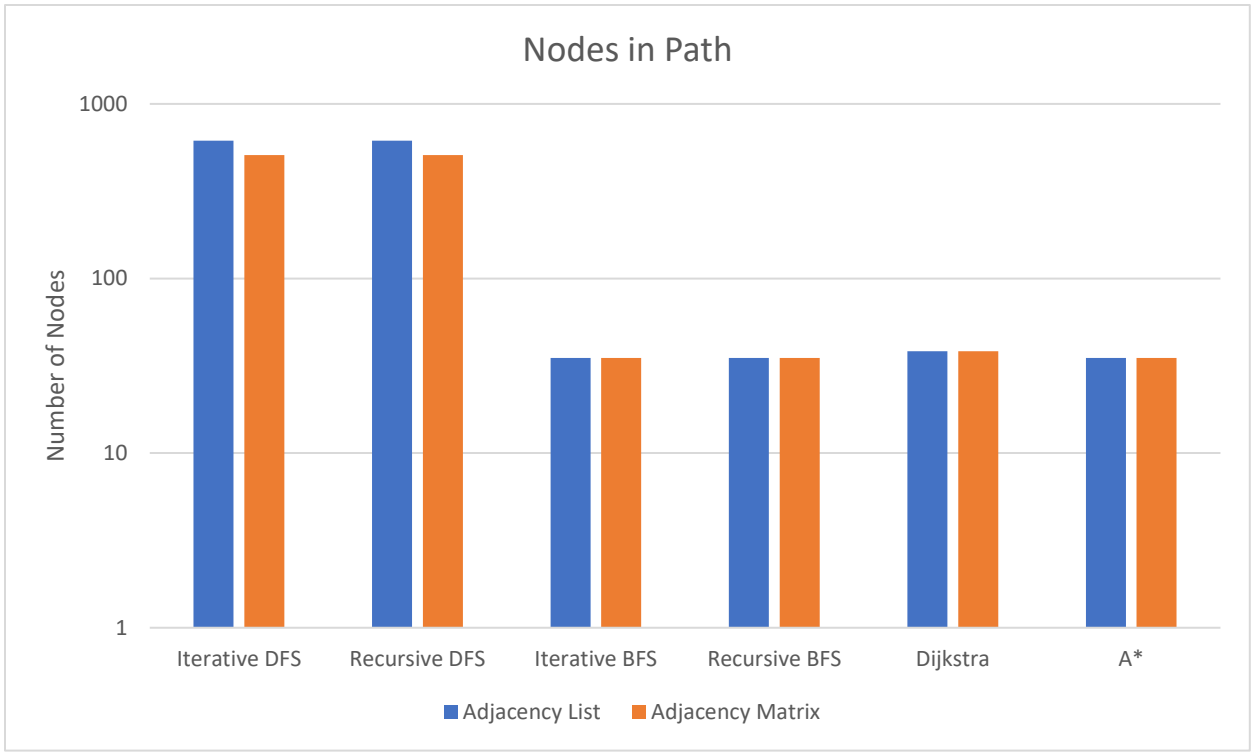
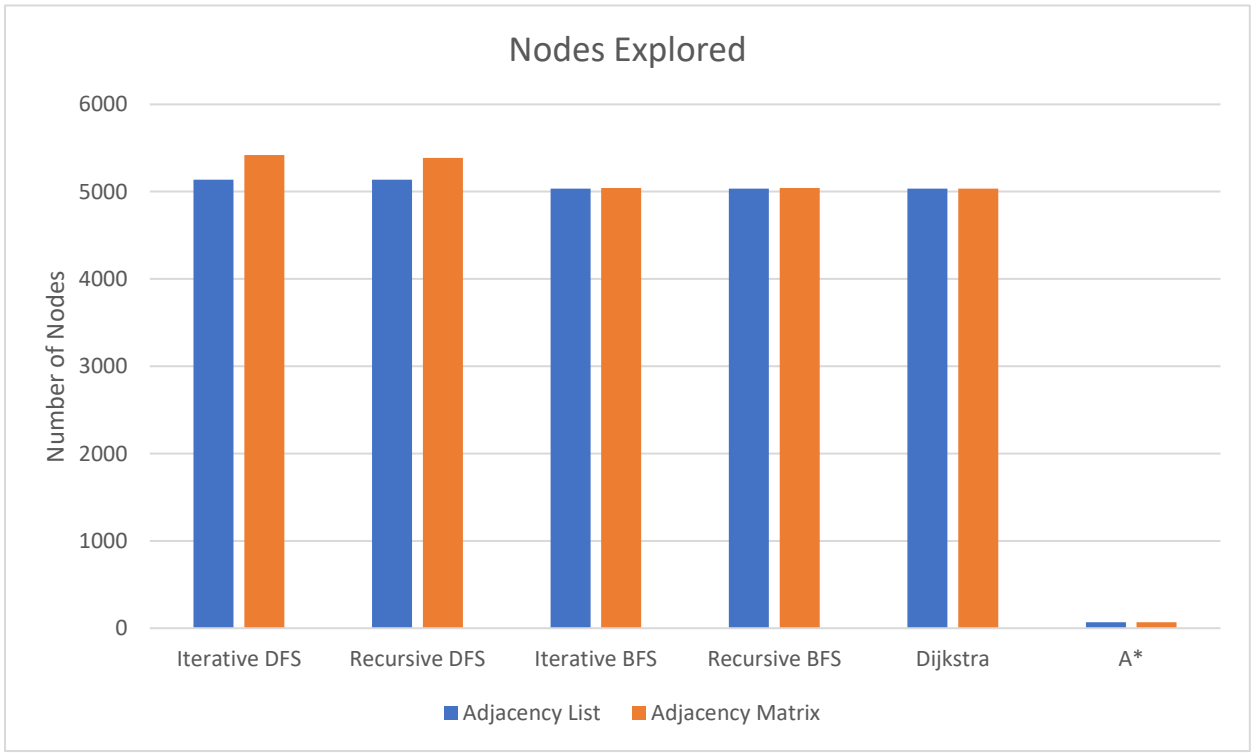
Nicholas Crothers

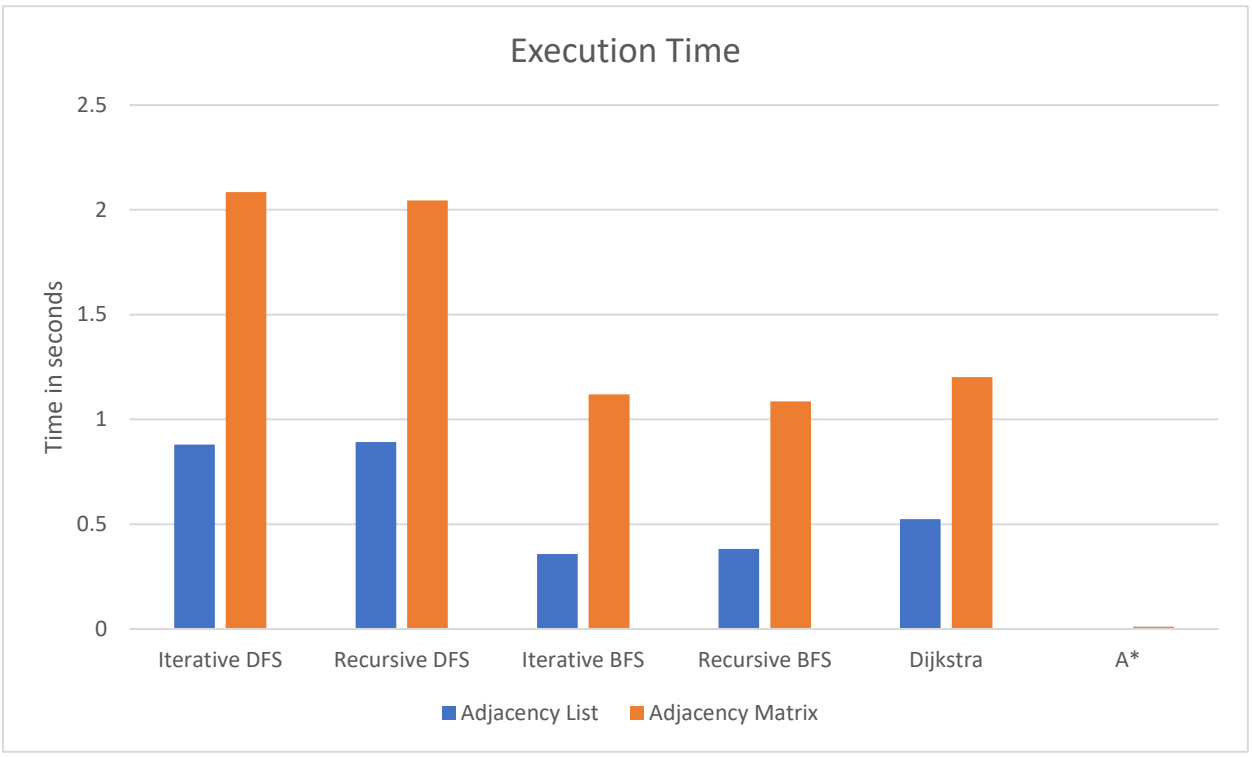
CSE 3353 Lab 2 Report

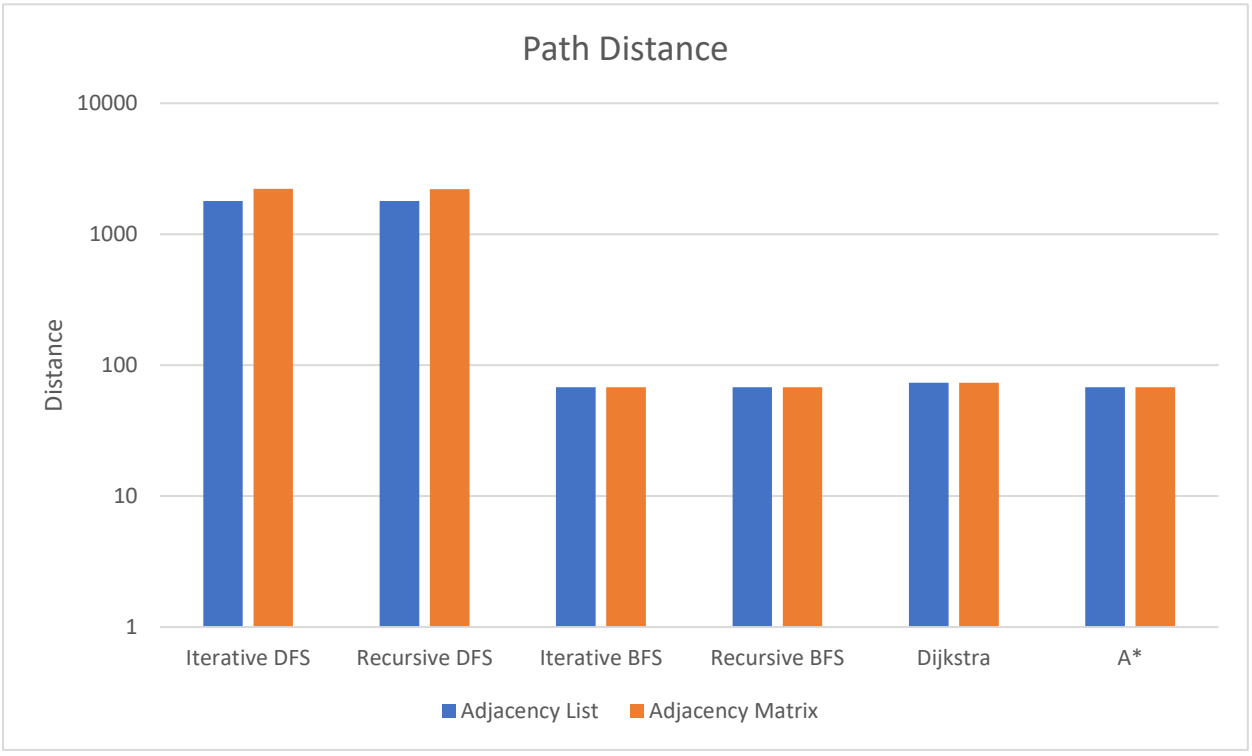
Adjacency List		Search Algorithm					
		DFS Iterative	DFS Recursive	BFS Iterative	BFS Recursive	Dijkstra	A*
Avg Normalized Results	Nodes in Path	1791.9	1791.9	68.8	68.8	74.58	68.8
	Nodes Explored	5137.82	5137.82	5035.42	5035.42	5033.9	68.8
	Execution Time	0.879058	0.891634	0.357758	0.381859	0.523576	0.002874
	Distance	1790.9	1790.9	67.8	67.8	73.58	67.8
	Cost	0	0	0	0	20.59721	30.62479

Adjacency Matrix		Search Algorithm					
		DFS Iterative	DFS Recursive	BFS Iterative	BFS Recursive	Dijkstra	A*
Avg Normalized Results	Nodes in Path	2217.02	2213.053	68.8	68.8	74.58	68.8
	Nodes Explored	5420.13	5385.968	5043.09	5043.09	5033.92	68.8
	Execution Time	2.085463	2.045739	1.119508	1.085355	1.201703	0.010889
	Distance	2216.02	2212.053	67.8	67.8	73.58	67.8
	Cost	0	0	0	0	20.59721	30.62045


Nodes in Path
1000
100
10
1
Number of Nodes
Iterative DFS
Recursive DFS
Iterative BFS
Recursive BFS
Dijkstra
A*
Adjacency List
Adjacency Matrix


Nodes Explored
6000
5000
4000
3000
2000
1000
0
Number of Nodes
Iterative DFS
Recursive DFS
Iterative BFS
Recursive BFS
Dijkstra
A*
Adjacency List
Adjacency Matrix


Execution Time
Time in seconds
2.5
2
1.5
1
0.5
0
Iterative DFS
Recursive DFS
Iterative BFS
Recursive BFS
Dijkstra
A*
Adjacency List
Adjacency Matrix


Path Distance
Distance
10000
1000
100
10
1
Iterative DFS
Recursive DFS
Iterative BFS
Recursive BFS
Dijkstra
A*
Adjacency List
Adjacency Matrix

Data Analysis:

First, when examining the difference between the adjacency list and the adjacency matrix, the adjacency matrix has a longer execution time than the adjacency list for each of the algorithms, by a large margin. This result ran counter to what I had thought; since the adjacency matrix has constant access to all the children, I had expected it to be faster. However, my algorithm pulls all the children of a specific node at once, so the impact that the access complexity has is greatly diminished by the getting of all children.

DFS produces substantially longer paths than the other algorithms. Because it goes deep into the graph then branches out, the path will likely be much longer than that of BFS, since BFS guarantees the shortest length path is found. DFS has the possibility of going through hundreds of nodes, then reaching the destination, even if the destination was adjacent to the start node. The execution time for DFS is likely much longer than the others despite having a similar count of visited nodes to BFS because it returns a much longer path than the other algorithms, and the time required to return the path increases as the path length increases. The difference between the iterative and recursive version is negligible, so the difference in performance between the two doesn't impact the overall performance of the algorithm as a whole.

BFS always finds a path that has the shortest length possible, because it searches in a radial pattern out from the start node, so the first time it hits the destination node, the shortest length path to it has been found. As with DFS, the difference in execution time between iterative and recursive BFS is negligible, so the different implementations don't affect the overall performance of the algorithm.

Dijkstra tends to find a path that is slightly longer than the shortest in terms of number of nodes, but it guarantees to find the path with the lowest cost. The number of nodes explored is

essentially equivalent to the number of nodes explored to BFS, and that is because it searches in a radial pattern – more or less – like BFS, but it prioritizes lower weight first. This is also why the total distance of the path is slightly longer compared to BFS. Because of the performance hit caused by the prioritization of lower weight, the execution time of Dijkstra is longer than BFS, despite the number of nodes visited being essentially the same.

A* was the algorithm that I believe was implemented slightly incorrectly. It always found the shortest path, with the lowest number of nodes visited by far, but the cost of the path was significantly higher than Dijkstra. I believe this is because I allowed the distance differential between nodes to affect the prioritization of nodes more than it should have. However, because of this, the execution time was insanely fast on the large dataset. The path found was shorter than Dijkstra, but it only took the most direct path it could to the node, thus why the number of nodes visited is always the same as the path.