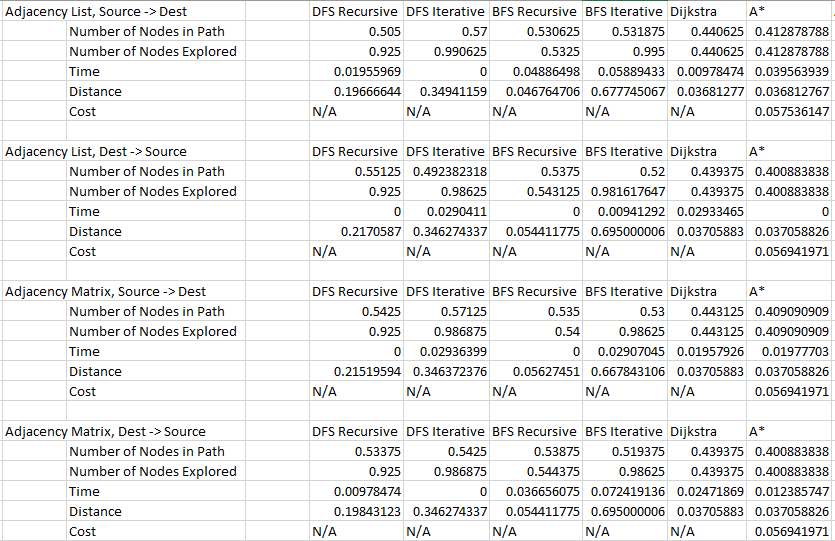
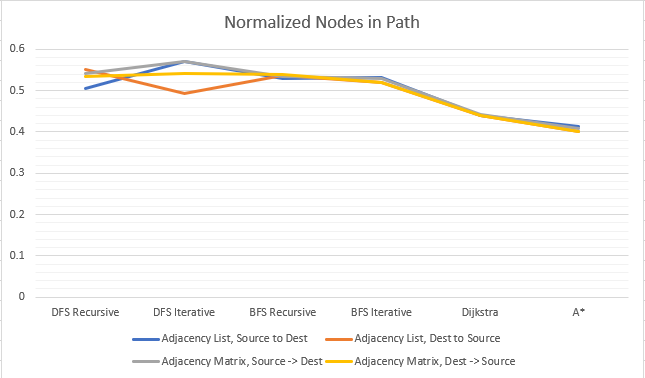
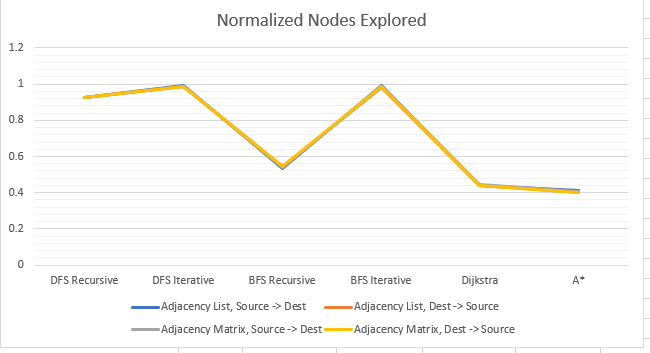
Andrew Breslauer

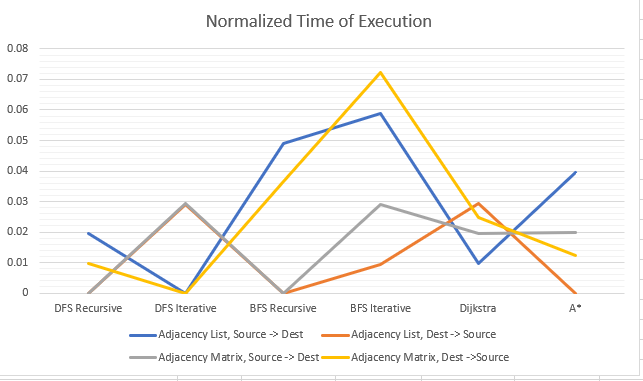




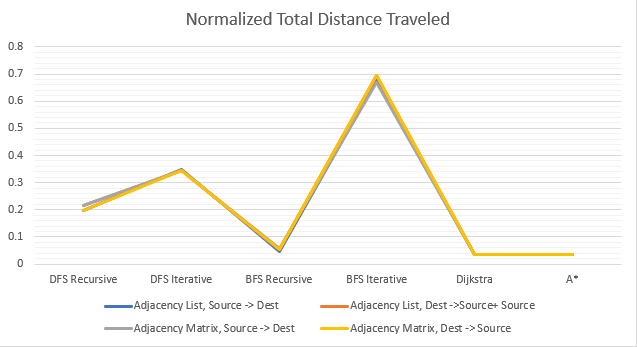
This graph shows that the number of explored nodes across each of the different search types is relatively the same. A\* and Dijkstra tended to find slightly shorter paths, but that makes sense because of the intelligent path-finding that is associated with them more closely than with DFS or BFS. The DFS iterative searching is the only type of search that seems to have any sort of major discrepancy among the Graph types and which order they were searched in, but that is most likely due to the pattern of randomization for each node, rather than the actual search algorithm.



The number of nodes explored for both A\* and Dijkstra are significantly lower than the number of nodes explored for BFS or DFS, which is exactly the intention of implementing the slightly more complicated algorithms. The smaller number of nodes explored is due to the faster pathfinding of the two algorithms, and the fact that they explore less nodes means that their execution time should be much faster.



This time of execution graph is incredibly mis-leading for a variety of reasons, but the main one can simply not be avoided with the tests being run on the 16-node graph: the path is found too quickly for the <chrono> timer to record any time, so the function returns 0 time of execution for most of the iterations. This graph would be much more helpful, as well as much more accurate to what the actual timing of the algorithms should be, if my graph structure would have worked on larger graphs and I had taken the time to execute them.



This graph shows the total distance travelled by each of the algorithms and the corresponding data structure. Unsurprisingly, Dijkstra and A\* had significantly lower relative distances travelled than either of the DFS types or BFS iterative; however, in surprising fashion, BFS recursive had almost as small of a distance travelled as A\* or Dijkstra. This, like the timing graph, is most likely due to the small size of the total graph and would most likely not be the same if larger graphs were used for testing.

To no one’s great surprise, the metrics that were recorded and contained meaning tended to heavily favor A\*, and then Dijkstra soon after, far more than BFS or DFS. A\* and Dijkstra do consider the slight cost of calculating the next optimal node to search, but on graphs of this size there is very little actual cost to the execution time. A\* always out-performed or equally as good as Dijkstra, and that is because the path taken by the A\* algorithm is the same path as a best-case Dijkstra search path, as well as best-case BFS and DFS. On the off-chance that, on a larger graph, the best-case BFS and/or DFS path was the correct path, there would still be significant timing differences, but instead in favor of BFS or DFS (whichever was favored) because they don’t have to account for calculating the optimal next node. Most of the data provided was inconsistent at best and would definitely be more accurate if ran on larger graphs (looking at the ones provided of 500,000 and 1,000,000 nodes, which are slightly larger than 16), but through the pure power of procrastination there wasn’t enough time left to fix my graph structure in order to accommodate larger graphs.

Last note: timing data in ouput.csv and normalizedOutput.csv may not match the data within normalizedOutput\_excel.xlsx, and that is mostly due to some last minute attempts to fix graph in order to accommodate the larger graphs and not disabling the saveData() function, which would then overwrite output.csv and normalizedOutput.csv.