Analysis of Performance of Two Implementations of Dijkstra's Algorithm

We wrote functions that created sparse, linear[1], and complete graphs and populated the weights with random numbers. Then the standard Dijkstra's algorithm and the improved Dijkstra's algorithm[2] were called on the graph for several vertices; the time to run the algorithm was recorded and repeated for several vertices, then the results were averaged to produce a time. The final results were saved to a file in CSV format. The files were imported into excel and graphs were made from the time data.

For the sparse graphs, implementation of the graph as either an adjacency list representation (ALR) or the adjacency matrix representation (AMR) did not seem to affect the time to complete the algorithms. However, the algorithm chosen did affect the time; the improved algorithm was at least twice as fast as the normal algorithm for less than 33,000 vertices.

For linear graphs, neither the implementation or the algorithm had much affect on time. The time was roughly linear with size. The improved algorithm was faster for small graphs but was the same as the standard algorithm for graphs with more than 1000 vertices.

For complete graphs, regardless of the implementation or algorithm, the time to complete the algorithm seemed to grow exponentially with graph size. Interestingly, the fastest algorithm was the standard algorithm on an AMR graph. The improved algorithms yielded roughly the same time regardless of graph implementation. The slowest was the standard algorithm on an ALR graph. The improved Dijkstra's algorithm was increasingly faster for ALR graphs and increasingly slower for AMR graphs.

[1] We define linear to mean a graph where node 1 is connected to node 2 and node n is connected to node n+1.

[2] Improved Dijkstra's algorithm is Dijkstra's algorithm implemented with a priority queue.