A5: Dijkstra’s Algorithm

5 May 2023
CSC2001F

VLNSHA004
Shaylin Velen

Goal of the experiment:

The goal of this experiment is to compare the theoretical bounds of Dijkstra's algorithm with the actual, measured performance of Dijkstra's shortest paths algorithm. We expect that the measured complexity, which was measured in the number of operations performed in Dijkstra's algorithm is bounded by the theoretical complexity of Dijkstra's algorithm which is Elog(V) where E and V are the number of Vertices and Edges in the graph, respectively. To elaborate, the actual number of processes counted will never be greater than the theoretical complexity of Dijkstra's algorithm.

Nature of experiment

The experiment was conducted in such a way that a graph was generated with a structured, set number of edges. Where the number of vertices take on the values (10, 20, 30, 40, 50) and the number of edges take on a range of values for each number of vertices. In total there are 25 graphs and the number of edges given a number of vertices take on the values in the range [(v(v-1))/m ; (v(v-1))] where v is the number of vertices. This was done to ensure that a range of different graph sizes and shapes were tested. The maximum value of the range is the maximum number of edges in a graph with v vertices before all vertices are fully connected. 5 numbers of edges were calculated as evenly spaced intervals over the range of the minimum and maximum number of edges for a given number of vertices.
The shape and edges of the graph were randomly generated. For example, if a graph had 20 vertices and 380 edges a list was created holding string values of Node1 – Node20 and 2 nodes would be randomly selected to generate an edge. To ensure that no graph had an edge that repeated itself a hash set data structure was used to only store unique edges.
A weight was generated randomly in the range 1 – 10 to ensure that no negative distances were assigned to edges. The hash set stored a list of type string as the data structure which were two randomly picked nodes: the source and destination and a weight assigned to this edge. A dataset was written into a text file, and then read into a graph using a map data structure to simulate a graph structure.
Dijkstra's algorithm was run over each graph generated where the algorithm looks to find the shortest path from the start node to all other nodes. The number of edge operations, number of vertex operations and the number of priority queue operations were counted while running Dijkstra's algorithm over the graph. A sum of these three values was then compared to the theoretical performance of Dijkstra's algorithm (ElogV).

Experiment design decisions:

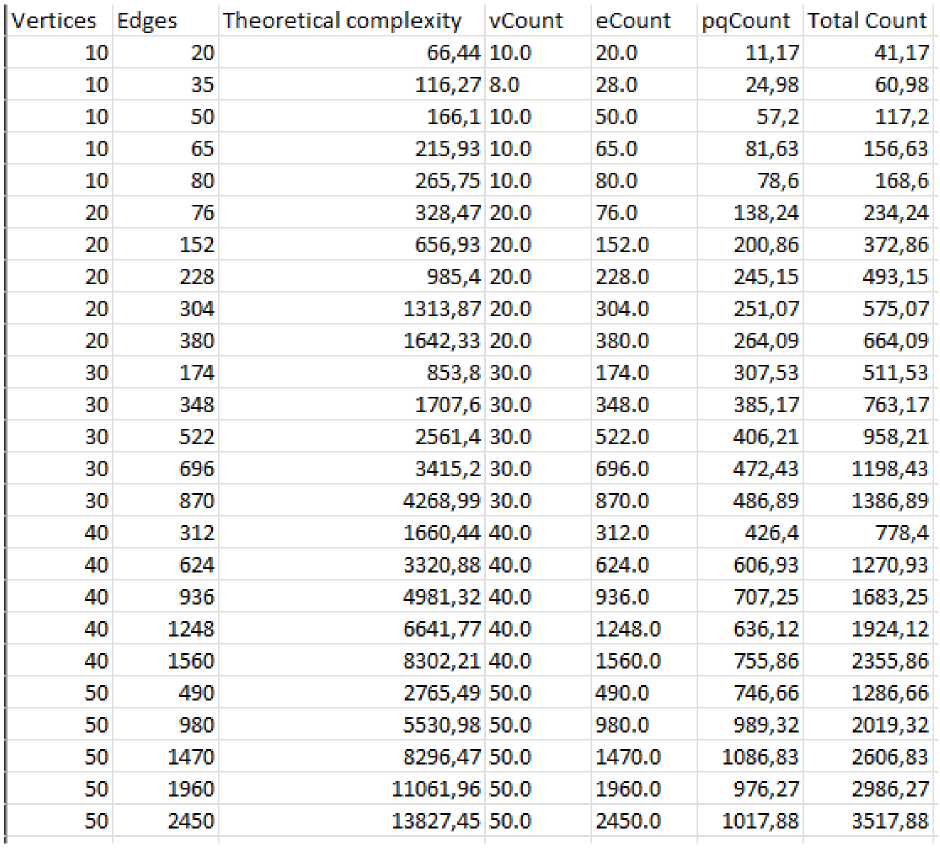
The edges generated were in the range 1-10 to ensure that no edge weights generated were negative, since Dijkstra's algorithm has inefficiencies when run over a graph with negative edges. Dijkstra's algorithm fails to find the shortest path to each node because it may ignore some paths that become shorter after traversing an edge.
The priority queue operations were added to the total operation count as these operations in the algorithm causes a higher complexity of the algorithm. The counter was incremented before an item was added to the priority queue and after an item was removed from the queue.
The experiment was carried out in a manner that ensures that Dijkstra's algorithm will always successfully count the shortest path from a start node to any other node in the graph, provided the node is reachable in the graph.
It is also worth noting that the graphs generated by the program do have nodes in the graph that are unreachable.

Design and implementation of OOP:

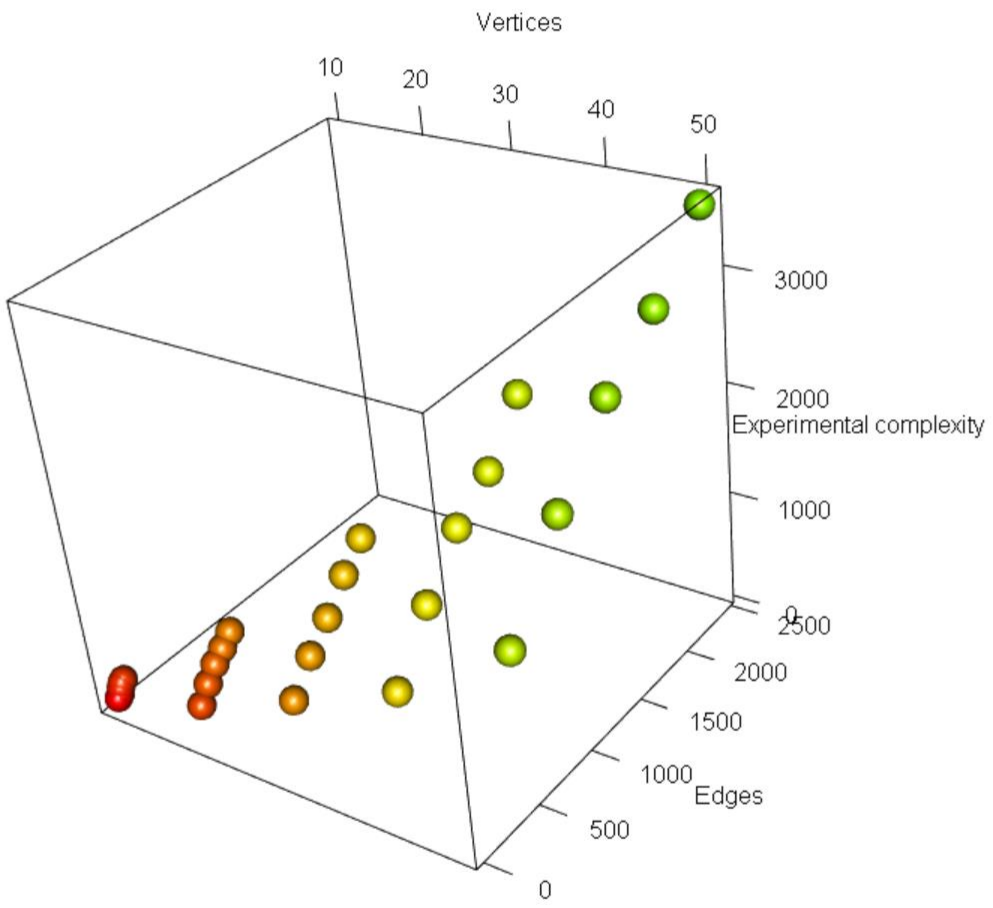
The classes created for the experiment include the given Graph, Vertex, Path and Edge classes as the structure and objects to create a graph and implement Dijkstra's algorithm. Other classes include: the GenerateGraph class, GraphExperiment class and the Graph Exception class used to throw an exception should an error be found in the running of Dijkstra's algorithm.

The generate graph class has a method to generate the data and store this generated data in a text file. Once this data has been generated, the graph experiment class holds the main method to run the program. When the program runs, an iterator is used to repeatedly call the GenerateGraph constructor which holds the program to generate a set of data, i.e., one graph holding say 20 vertices and 380 edges. Once a graph is generated the processRequest method is called from the Graph class. This method has been modified to call the Dijkstra method, store the counted values in the algorithm and then output these values to a text file and a .csv file.

Final results:



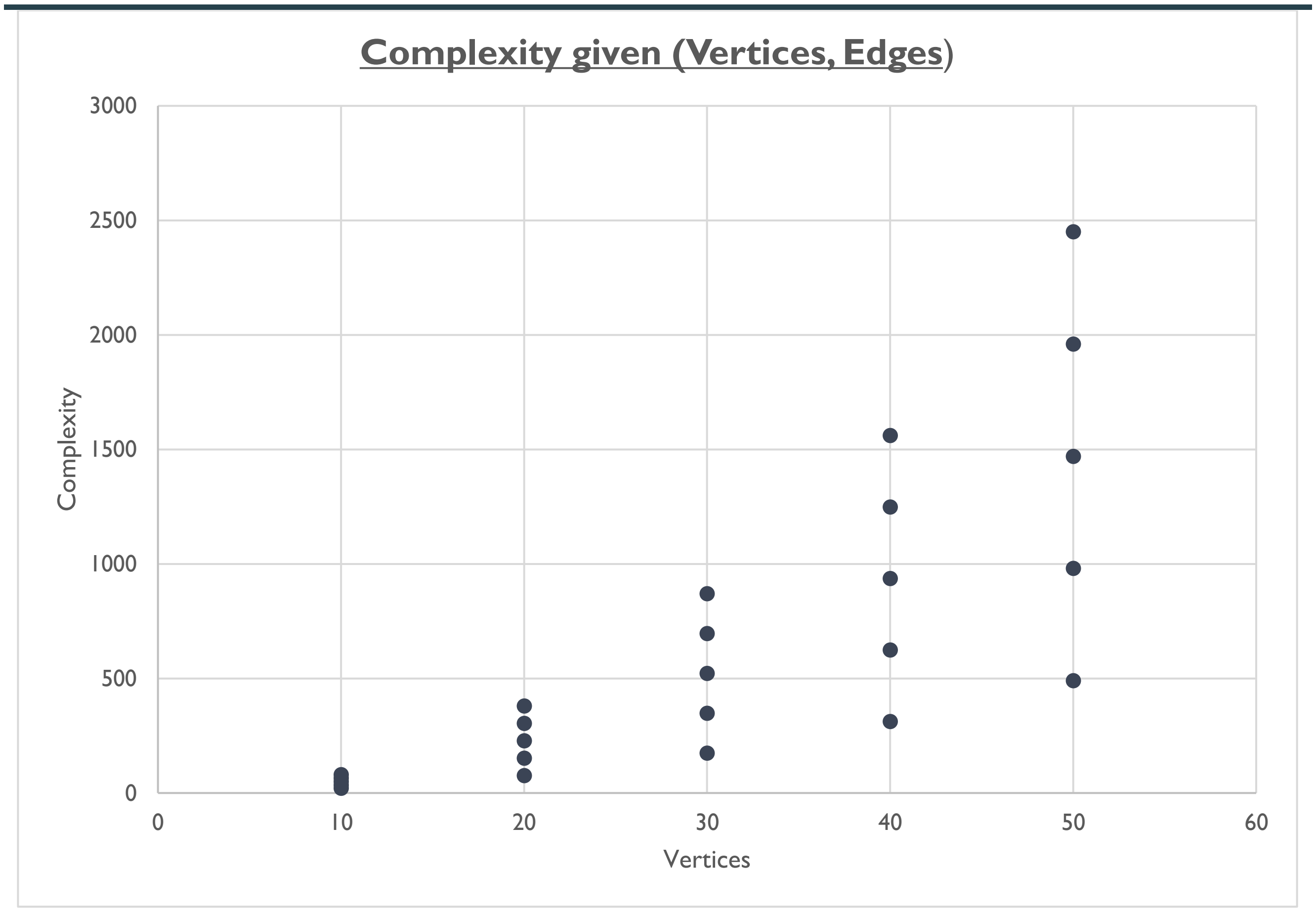
Vertices	Edges	Theoretical complexity	vCount	eCount	pqCount	Total Count
10	20	66,44	10.0	20.0	11,17	41,17
10	35	116,27	8.0	28.0	24,98	60,98
10	50	166,1	10.0	50.0	57,2	117,2
10	65	215,93	10.0	65.0	81,63	156,63
10	80	265,75	10.0	80.0	78,6	168,6
20	76	328,47	20.0	76.0	138,24	234,24
20	152	656,93	20.0	152.0	200,86	372,86
20	228	985,4	20.0	228.0	245,15	493,15
20	304	1313,87	20.0	304.0	251,07	575,07
20	380	1642,33	20.0	380.0	264,09	664,09
30	174	853,8	30.0	174.0	307,53	511,53
30	348	1707,6	30.0	348.0	385,17	763,17
30	522	2561,4	30.0	522.0	406,21	958,21
30	696	3415,2	30.0	696.0	472,43	1198,43
30	870	4268,99	30.0	870.0	486,89	1386,89
40	312	1660,44	40.0	312.0	426,4	778,4
40	624	3320,88	40.0	624.0	606,93	1270,93
40	936	4981,32	40.0	936.0	707,25	1683,25
40	1248	6641,77	40.0	1248.0	636,12	1924,12
40	1560	8302,21	40.0	1560.0	755,86	2355,86
50	490	2765,49	50.0	490.0	746,66	1286,66
50	980	5530,98	50.0	980.0	989,32	2019,32
50	1470	8296,47	50.0	1470.0	1086,83	2606,83
50	1960	11061,96	50.0	1960.0	976,27	2986,27
50	2450	13827,45	50.0	2450.0	1017,88	3517,88



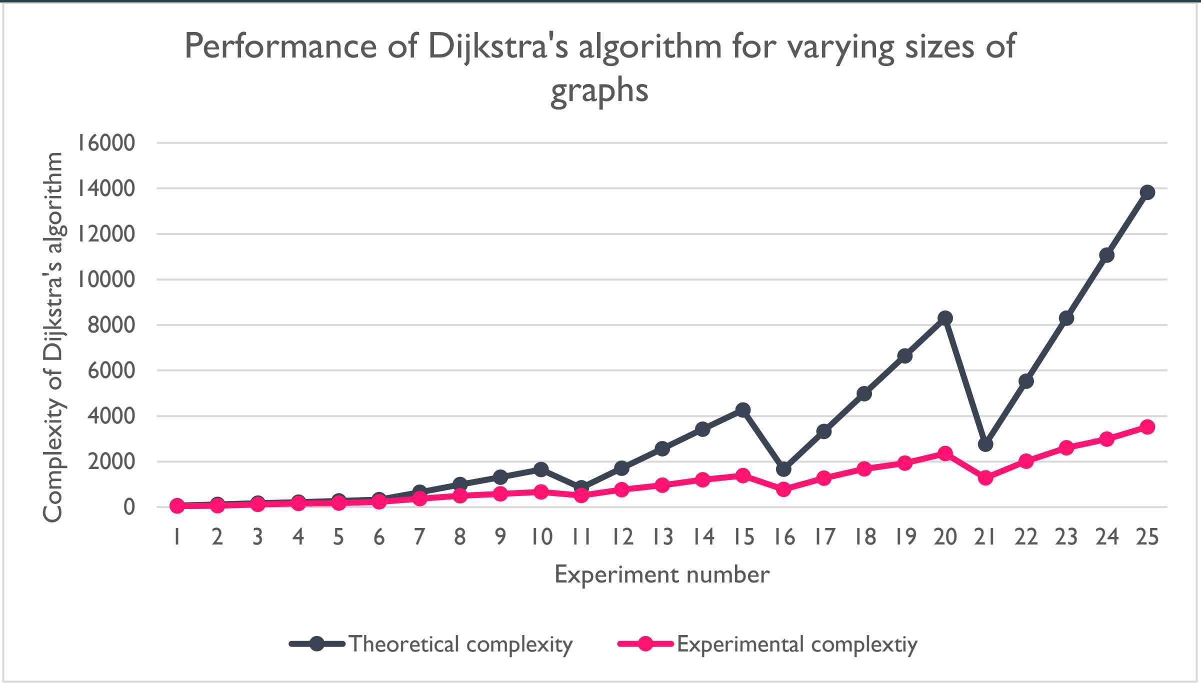
Plot A: Experimental complexity given the size and density of the graph

Plot A represents the number of vertices and edges of any given graph in the experiment plotted against the experimental complexity given that size and shape of the graph. This would measure the performance of Dijkstra's algorithm given a graph size that would be sparse, dense, large or small.

The resulting plot depicts that the larger the graph gets the higher the complexity of Dijkstra's algorithm. The size and shape of the graph are large determining factors of the performance of the algorithm.



Plot B
The plot depicts that for a given number of vertices and edges, the experimental complexity increases. The larger the size of the graph, the higher the experimental complexity measured was. This is plotted using a scatter plot showing varying combinations of vertices and edges, which provide the complexity of graphs with 25 different sizes and densities.

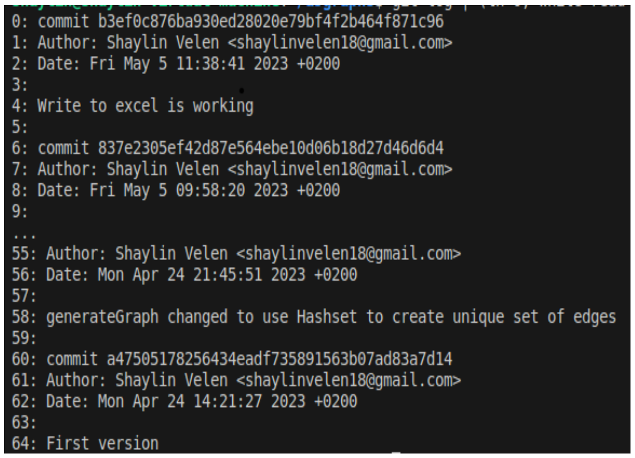


Plot B
The graph depicts the theoretical and experimental performance of Dijkstra's algorithm. For each experiment number 1 -25 a number of vertices and edges was chosen to create graphs of varying shapes. The graphs show that with lower vertices, for example experiment 1-10, that the experimental and theoretical performance of the algorithm are sufficiently close to each other, and that the theoretical performance is a good estimator of the complexity of the algorithm. However, it is shown that graphs with a higher number of vertices have a significantly lower complexity, than that of the theoretical complexity.
One of the key takeaways of the experiment was that the experimental performance is never greater than the theoretical performance of the algorithm and that the theoretical complexity is in fact a bound for the experimental performance of Dijkstra's algorithm.

Creativity:

The assignment brief stated that the program was meant to either output, the results of an experiment to a text file or the screen when running the program. The implementation of the GraphExperiment program allows for the results of the experiment to be outputted to an excel file where results are easier to read and interpret.
A 3d scatter plot was shown to depict the performance of Dijkstra's algorithm more accurately against the size and shape of the graph.

Git log:



```
0: commit b3ef0c876ba930ed28020e79bf4f2b464f871c96
1: Author: Shaylin Velen <shaylinvelen18@gmail.com>
2: Date: Fri May 5 11:38:41 2023 +0200
3:
4: Write to excel is working
5:
6: commit 837e2305ef42d87e564ebe10d06b18d27d46d6d4
7: Author: Shaylin Velen <shaylinvelen18@gmail.com>
8: Date: Fri May 5 09:58:20 2023 +0200
9:
...
55: Author: Shaylin Velen <shaylinvelen18@gmail.com>
56: Date: Mon Apr 24 21:45:51 2023 +0200
57:
58: generateGraph changed to use Hashset to create unique set of edges
59:
60: commit a47505178256434eadf735891563b07ad83a7d14
61: Author: Shaylin Velen <shaylinvelen18@gmail.com>
62: Date: Mon Apr 24 14:21:27 2023 +0200
63:
64: First version
```