Assignment 5
Dijkstra's Algorithm Experiment

BY WTBROC002

Roche Witbooi | CSC2001F | 05 May, 2023

Contents

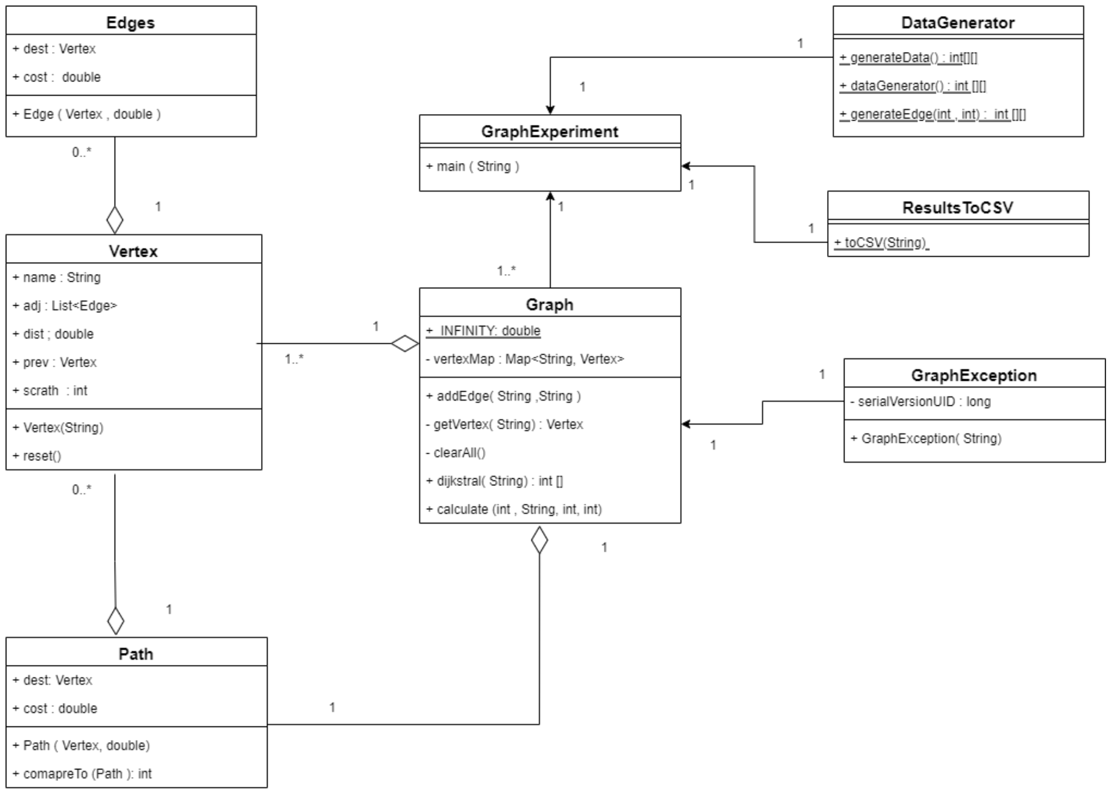
Object-Oriented Design:

The main program is run from the Graph experiment class, which uses static methods from the DataGenerator class to create various data sets required for the experiment. This allows the data-generating process for vertex and edges as well as the node edge-generating process to be encapsulated and separated from the main Experiment.

Graphs are the data structure required for the experiment since each test requires for a unique graph to be created. These graphs are created in the graph class which encapsulates reading in the data file containing the graph edged, creating the graph, and running Dijkstra's algorithm. TheGraphException class encapsulates how errors are handled when the input text file does comply with the requirements for generating a Graph.

A graph consists of vertices and edges, which are represented by instances of Vertex and Edge classes. These classes store the attributes of the graph components and are used to construct the graph on which Dijkstra's algorithm is run. The path class is used to store the Vertex name and the weight of previously visited vertex in Dijkstra's algorithm.

Finally, the ResultsToCSV file encapsulates the method used to convert the results from a text file format to CSV which streamlines the processes of visually representing the data.



Experiment Goal:

The main goal of the experiment is to demonstrate that the complexity of Dijkstra's algorithm is O (E log (V)). This means that this experiment will calculate the number of edge and vertex processes that occur when Dijkstra's algorithm is run on graphs with varying numbers of vertices and edges. This calculation will then be compared to the theoretical value produced by complexity = E log V to demonstrate that O (E log (V)) is a suitable upper bound estimate for the complexity of running this algorithm on any Graph.

For this experiment all the graphs are directed, weighted graphs which are required by Dijkstra's algorithm. Furthermore, the edges that the graphs consist of have been randomly generated to eliminate bias and ensure that experiment is as accurate as possible. The edges in the graph must be unique and cannot start and end at the same vertex.

The calculation of the experimental complexity (the complexity determined by counting the number of processes) is done by instrumentation. Counters are instantiated for the number of vertex, edge, and priority queue processes before Dijkstra's algorithm is run. Whilst the algorithm is being processed the vertex counter increases when a vertex is processed, the edge counter is incremented when an edge is processed, and the priority queue counter is incremented when an edge is added and removed from the priority queue. Intuitively, complexity is the measure of the amount of time the computer takes to run an algorithm. Since processes increase the time taken to run an algorithm summing the counting of each type of process (vertex, edge, and priority queue processes) produces a more specific estimate of the complexity. However, since adding and removing an edge to a priority queue's time differs from that of processing an edge and vertex, the value of priority queue operations is adjusted to the log to the base 2 of the size of the priority queue.

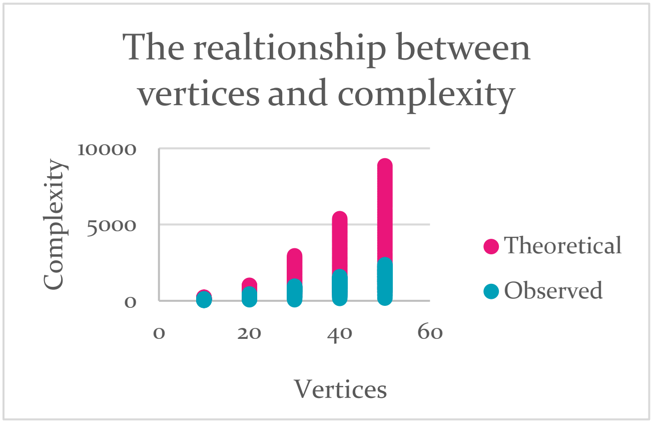
The test is performed by generating graph with a given number of vertices and edges. The edges are generated randomly and printed to a text file. These random edges are then used to create many graphs.

The program then runs Dijkstra's algorithm to determine on the graph and the instrumentation is used to determine the actual number of operations that are performed whilst the algorithm is being run. The values of the instrumentation counters are printed to a results text file which is written to a csv file to perform analysis in these output values.

The tests are repeated multiple times during the experiment to generate enough data points to draw a conclusion for the experiment.

Results

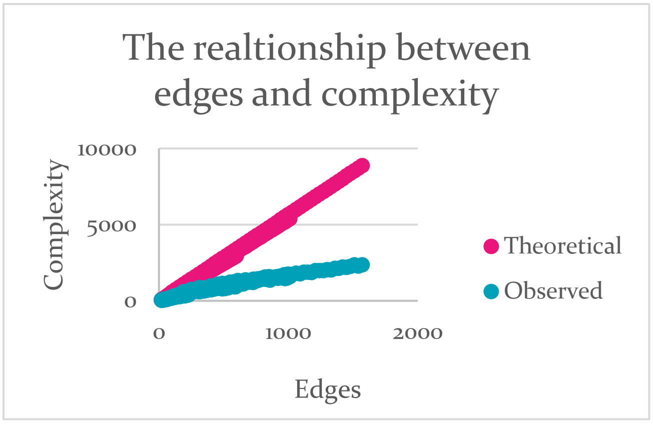
In an experiment with 250 tests were run, where each test had a unique graph with vertex ranging from 10 -50 and the number of edges ranging from 10% to 80% of the maximum number of edges the graph could possibly have, the following was observed. The intervals between edge values were held constant.



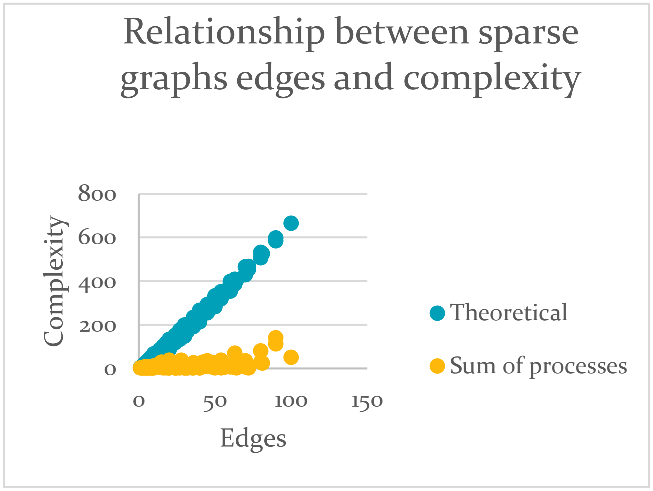
Graph 1:

The observed complexity increased, and the number of vertices increased. The increase in observed number of processes appears to be proportional to the number of vertices in each graph. The graph depicts this by the pink being plotted at higher complexity values than the blue values.

Graph 2:

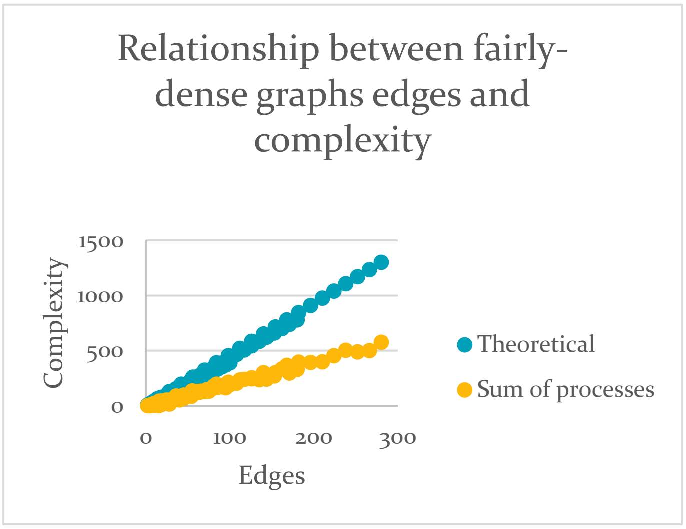


Similarly, as the number of edges in the graphs increases, so does the observed complexity. This implies that the complexity of the algorithm is also proportional to the number of edges in the graph. The graph depicts this by the pink being plotted at higher complexity values than the blue values.



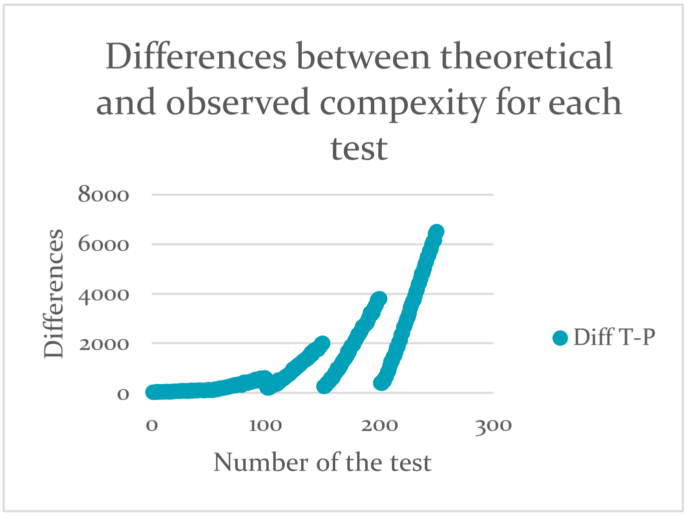
Graph 3:

In the case where the graphs were sparse meaning the number of edges were fewer than the number of vertices the Sum of processes were found to be substantially less that the theoretical number of processes.



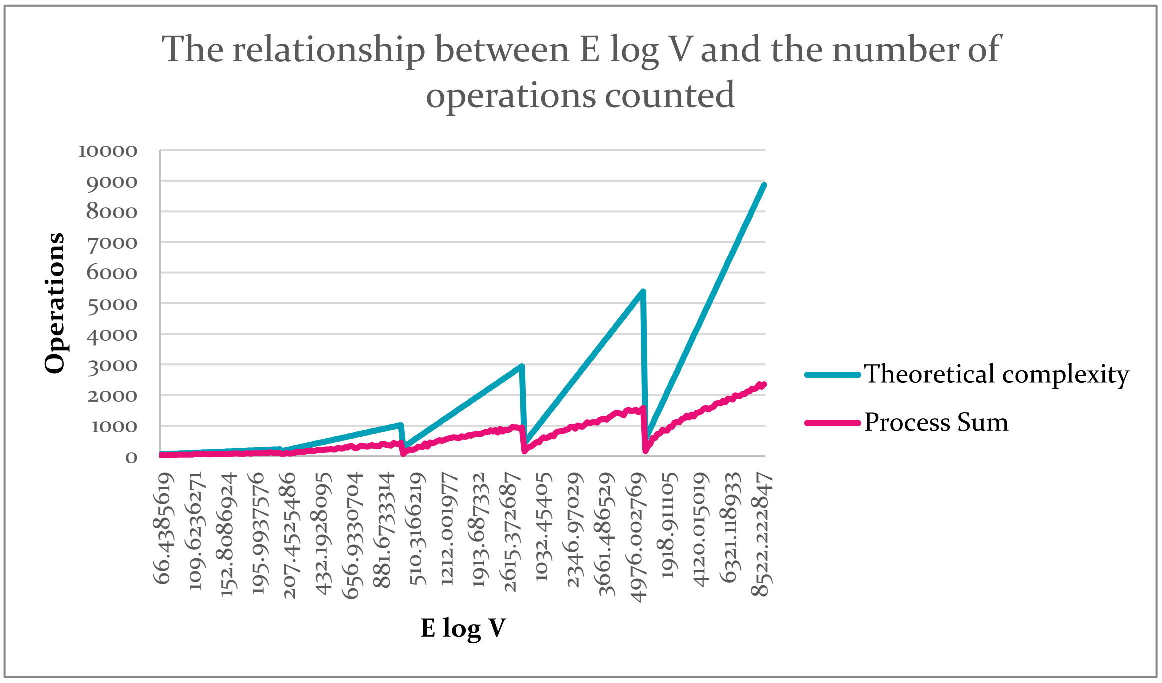
Graph 4:

Where the number of edges were greater than the number of vertices the values of theoretical and observed complexity were closer in value. However, despite being closer to the theoretical bound the observed sum of processes do not exceed the theoretical bound values for any of the graphs.



Graph 5:

However, across all tests the differences between the theoretically estimated complexity and the observed complexity were always observed as positive. Throughout the experiment all the test produced observed process sums which were less than the estimate provided by E log V.



In the graph above the observed number of operations were plotted against the E log V value for each graph where E represents the total number of edges in the graph and V represents the total number vertices in the graph for a given test.

From looking at the graph it is evident that the number of operations that were observed are constantly less than or equal to the E log V values. This is evident as the pink line is never above the blue line in the graph across all tests performed.

Interpretation of results

From Graphs 1 and 2 it is evident the number of operations performed by Dijkstra's algorithm to determine the shortest path in a graph is dependent on the number of vertices and edges in the graph. The more edges and vertices in the graph imply more operations need to be performed to find the shortest path. This means that the order of the algorithm includes the value of vertices and edges. Furthermore, these graphs illustrate that even for the 5 values of vertices and varying number of edges the observed values were found to be smaller than or equal to the theoretical bound.

Graphs 4 and 5 illustrate the effect that graph density has on the number of operations in Dijkstra algorithm. For graphs with fewer edges the sum of processes or observed complexity is likely far less than the theoretical complexity. This is because fewer edges in a graph leads to more vertices not being connected to the graph structure, thus minimizing the number of operations the algorithm requires. Conversely more edges mean more edges and vertices must be processed by the algorithm which is seen in Graph

5. However, similarly to the first two graphs, Graphs 4 and 5 consolidate the notion that the value of the observed processed is observed to be less than the E log V value. Since the experiment was conducted on more dense graphs it can be inferred that the notion of E log V being theoretical bound for the number of operations performed by Dijkstra's Algorithm holds true for sparse graphs.

To confirm this observation that the observed operation values do not exceed the theoretical bounds, the differences between the theoretical and observed number of operations were plotted as seen in Graph 5. For all 250 tests with randomly generated graphs, the difference was found to be greater than or equal to zero. This confirms the notion that the actual operations performed by Dijkstra's algorithm does not exceed the value represented by multiplying the total number of edges in the graph by the log to base 2 of the number total number of vertices in the graph. It is evident that E log V is a valid upper bound estimate for the complexity of Dijkstra's Algorithm.

Creativity:

The main creative aspects of this experiment involve automating the raw data generation process. Raw data refers to the two-dimensional array storing vertex and edge values. The first index in the array represents the number of vertices in the graph and the remaining array elements specify the number of edges in each graph.

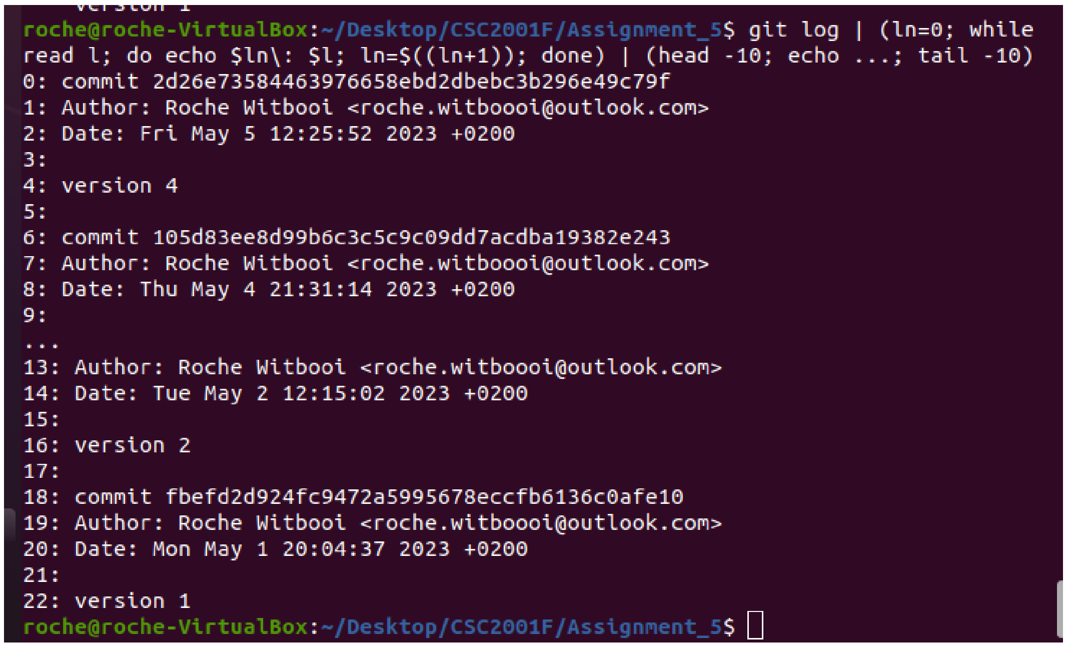
Additional code was written in the DataGenerator class to automate the generation of this data to create various graphs. The dataGenerator() method requires the programmer to manually specify the number of vertex sets, the interval between vertex sets, the number of edge sets, and the graph density type. The program then takes these values and generates a 2D array with the values corresponding to selections made by the user.

This function extends the utility of the program as the user can generate large data sets easily. This is an essential part of experimentation as replication of an experiment makes trends in the output more identifiable which strengthens the argument made for the conclusion.

Furthermore, the user's ability to specify the type of graph they are interested in allows the user to further assess the relationship between graph density and complexity. The types of graphs are (1) sparse, (2) fairly dense, and (3) dense graphs have an impact on how close the experimental complexity comes to the theoretical bound. This function allows for further exploration of this relationship without much effort from the users. This function allows the experiment conductor to look at the effect the graph density has on complexity. As graphs can be generated graphs with varying degrees of density based on the type selection. These graph types include sparse, where the number of edges is less than the number of vertices, fairly dense which has edge numbers ranging from 10%-50% of the maximum number of edges (which is v^2 – v since edges form a vertex to itself are not permitted). Furthermore, the experiment also considers dense graphs where the edges are 50% -90% of the maximum number of edges.

Another creative component of this program is writing the results of each test into a CSV file. This is done in the ResultsToCSV class and streamlines the processes of analyzing the output provided by running the experiment. The class reads the results text file and writes each line into the CSV file while separating the values into a cell. By automating this aspect of the experiment human error is avoided since the output data does not have to be retyped. This function allows for more reliable results and elevates the experiment conductor from the tedious process of retyping many data points.

Git log:



```
roche@roche-VirtualBox:~/Desktop/CSC2001F/Assignment_5$ git log | (ln=0; while
read l; do echo $ln\: $l; ln=$((ln+1)); done) | (head -10; echo ...; tail -10)
0: commit 2d26e73584463976658ebd2dbebc3b296e49c79f
1: Author: Roche Witbooi <roche.witboooi@outlook.com>
2: Date: Fri May 5 12:25:52 2023 +0200
3:
4: version 4
5:
6: commit 105d83ee8d99b6c3c5c9c09dd7acdba19382e243
7: Author: Roche Witbooi <roche.witboooi@outlook.com>
8: Date: Thu May 4 21:31:14 2023 +0200
9:
...
13: Author: Roche Witbooi <roche.witboooi@outlook.com>
14: Date: Tue May 2 12:15:02 2023 +0200
15:
16: version 2
17:
18: commit fbefd2d924fc9472a5995678eccfb6136c0afe10
19: Author: Roche Witbooi <roche.witboooi@outlook.com>
20: Date: Mon May 1 20:04:37 2023 +0200
21:
22: version 1
roche@roche-VirtualBox:~/Desktop/CSC2001F/Assignment_5$
```