Zwivhuya Ndou

NDXWI002

CSC2001F ASSIGNMENT 5

ELECTRICAL AND COMPUTER ENGINEERING

I, NDXZWI002, hereby declare that this assignment is my original work and that all sources have been acknowledged. I understand that plagiarism is the presentation of another person's work or idea as my own, without proper referencing, and that it is a serious offence. I declare that I have read and understood the plagiarism policy of my institution and that this assignment complies with those requirements.

Content:

1. OO Design

Classes:

makeGraphDataset.class, FunctionTest.class, test.class

Interaction:

makeGraphDataset.class

The makeGraphDataset class randomly generates edges using the createEdges() method and use the writeToFile(int, int) method to create a text file of graphs with predefined lists of number of vertices and edges.

FunctionTest.class

The program starts by reading a text file using the addFileEdgesToQueue() method. This method creates a Graph object to which edges from the text file are added using the addEdge(String, String, double) method from the Graph class. This step ensures that the Graph object contains all the necessary information to apply Dijkstra's algorithm.

The makeRequests(int, int, int) method is then used to randomly select a start vertex and a destination vertex in each graph text file. The Dijkstra algorithm is then used to find the shortest path from the chosen start vertex to the chosen destination vertex. This process is repeated for each graph file.During the execution of the Dijkstra's algorithm, the program counts the number of operations required to find the shortest path. These operations, together with the number of edges and vertices for that operation, are stored in a CSV file for further analysis.

The test.class runs the main function, which implements the entire program to generate the dataset and test the Dijkstra algorithm. The main function is responsible for coordinating the steps involved in reading the text files, running the Dijkstra algorithm, and storing the results in a CSV file for further analysis. This process ensures that the Dijkstra algorithm is evaluated thoroughly using a range of input data and provides a reliable estimate of its performance and efficiency.

test.class:

Runs the main function that implements the entire program to generate the dataset and test the dijkstra’s algorithm.

2. Introduction

The Dijkstra algorithm is an algorithm for solving the shortest path problem, which involves finding the shortest path between nodes in a graph. The performance of the algorithm is critical in practical applications, especially when dealing with large and complex graphs. The worst-case time complexity of the algorithm is proportional to E*log|V|, where E and V represent the number of edges and vertices in the graph, respectively.

To evaluate the efficiency and scalability of the algorithm, it is essential to understand the relationship between its theoretical performance and its actual performance when applied to specific inputs. This relationship is illustrated in Figure 4, which shows two plots: the theoretical plot and the operations plot.

3. Experiment

Goal

Testing Dijkstra's algorithm aims to evaluate its efficiency and suitability for solving the shortest path problem in various applications by analyzing its worst-case time complexity and actual performance. If the algorithm meets the requirements of the application, it would be considered a viable solution, otherwise, modifications or alternative algorithms may need to be explored.

Methodology

The method involved running the Dijkstra algorithm on different input graphs with various vertices and edges sizes, measuring the performance in terms of number of operations made to compute the shortest path. The worst-case time complexity, E*log|V|, would also be analyzed and compared to the algorithm's actual performance to evaluate its scalability and efficiency.

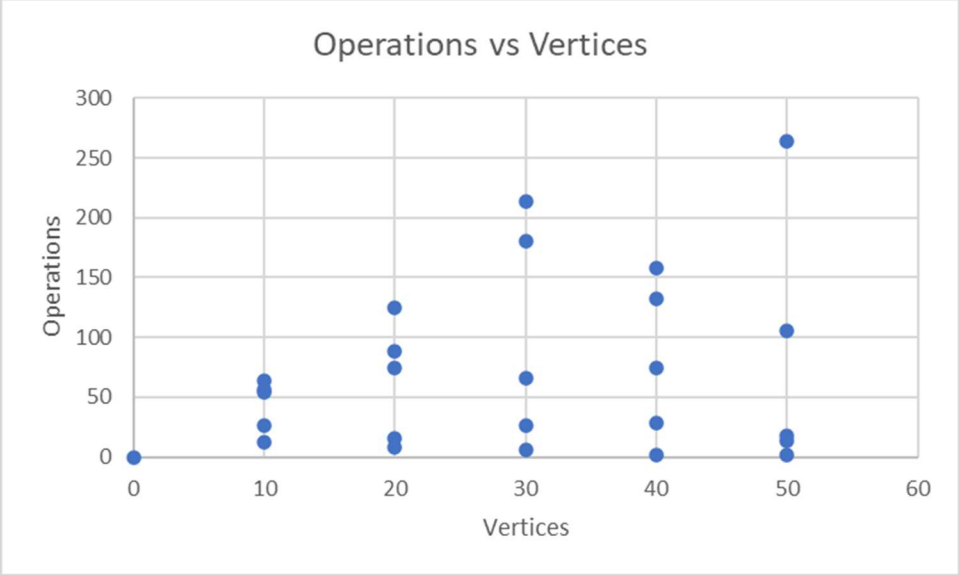
In the Dijkstra's algorithm, the size of the priority queue is dynamic and changes as vertices are added or removed. The variable pqCounter is used to keep track of the current size of the priority queue. The pqCounter is incremented each time a vertex is added or removed from the priority queue.

To count the number of operations, the pqCounter uses the logarithmic equation log(pq.size())/log(2). This equation calculates the logarithm of the current size of the priority queue, divided by the logarithm of 2. This calculation results in the number of operations required to execute Dijkstra's algorithm.

By using pqCounter and the logarithmic equation, the Dijkstra's algorithm's performance can be evaluated, and its efficiency can be analyzed in terms of the number of operations required to compute the shortest path.

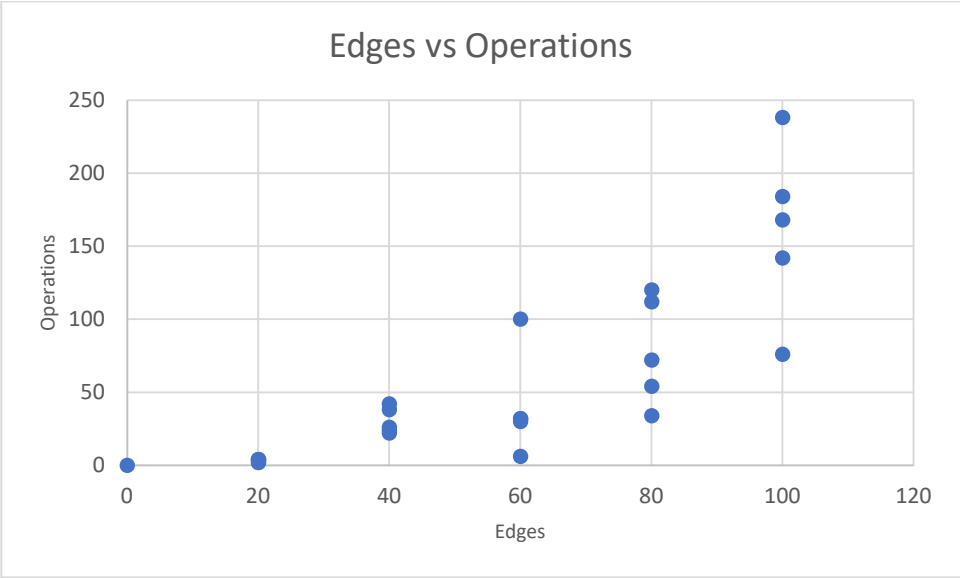
4. Results

4.1 Vertices vs Operations



When comparing the number of operations with the number of vertices, we can evaluate the algorithm's scalability. If the number of operations required to compute the shortest path increases linearly with the number of vertices, then the algorithm can be considered scalable but from figure 1 not much can be deduced from the vertices' plot above due to the number of data points used.

4.2 Edges vs Operations



In Figure 2, we can see a scatter plot with the number of edges and the number of operations required to execute Dijkstra's algorithm for each graph. The plot's independent variable, which is the number of edges, is on the x-axis, while the dependent variable, the number of operations, is on the y-axis.

As the number of edges in the graphs increases, the scatter plot points are initially concentrated but gradually spread out. This means that the number of operations required to execute Dijkstra's algorithm also increases with more edges.

The increasing trend indicates that the larger the graph, the more operations are needed to compute the shortest path using Dijkstra's algorithm. This makes sense because the algorithm needs to evaluate more vertices and edges in larger graphs to find the shortest path. Thus, this trend highlights the importance of considering the size of the input graph when evaluating the performance and efficiency of Dijkstra's algorithm.

4.3 Operations vs Theoretical

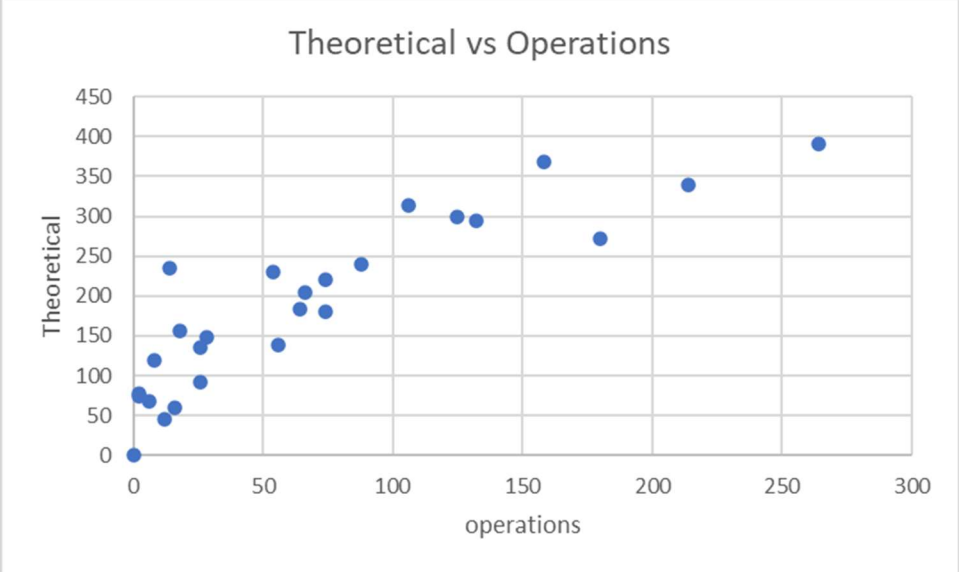


Figure 3

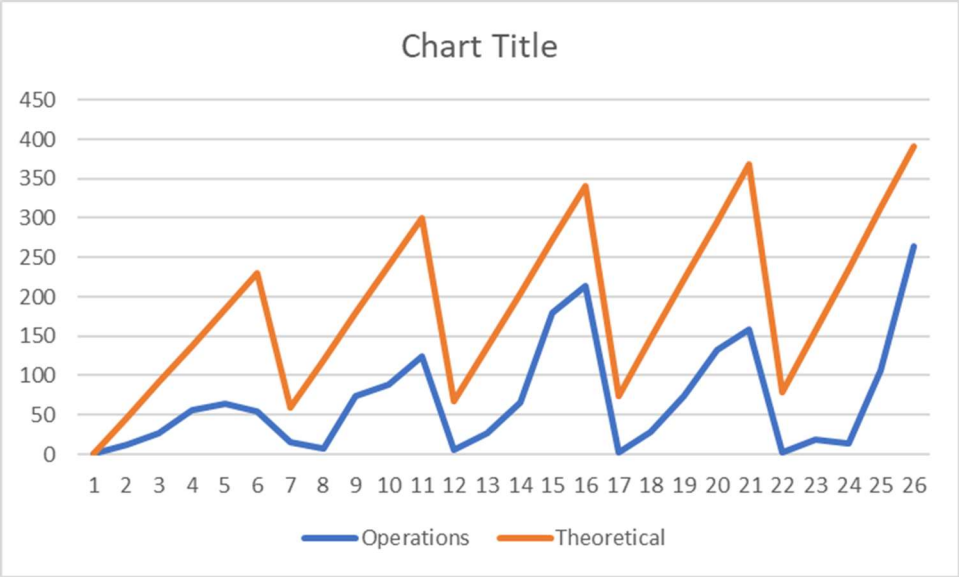


Figure 4

In figure 3, The results of a scatter plot between operations and theoretical in a Dijkstra algorithm would show how well the algorithm performs in terms of time complexity. If the scatter plot points fall close to the theoretical line, it indicates that the algorithm is performing efficiently and within the expected time complexity.

The plot in Figure 4 illustrates the relationship between the theoretical and operations performance of the Dijkstra algorithm. The theoretical plot represents the upper bound of the algorithm's performance, which is given by the E*log|V| function. On the other hand, the operations plot shows the actual performance of the algorithm in terms of the number of operations required to execute it for a given input.

Based on the plot, we can see that the operations plot is always below or equal to the theoretical plot, indicating that the algorithm is indeed bounded by the E*log|V| function. Additionally, we can observe that the operations plot increases at a rate that is less than or equal to the theoretical plot, which confirms that the algorithm's performance is indeed within the theoretical upper bound.

In summary, the plot in Figure 4 provides evidence that the Dijkstra algorithm has a worst-case time complexity of E*log|V| and that the algorithm's actual performance is always within this upper bound. This information is useful in understanding the efficiency of the algorithm and in comparing it with other algorithms for solving similar problems.

5. Limitations

Randomly generating datasets of graphs is a useful approach to test Dijkstra's algorithm, but it has limitations. The generated graphs may not cover all possible graph types or real-world phenomena, resulting in a limited sample size and potentially biased

conclusions. Additionally, it may be difficult to control the properties of the graphs being generated, which limits the ability to test the algorithm's performance under specific conditions. Therefore, while useful, randomly generated datasets should be used in conjunction with other testing methods to obtain a more comprehensive evaluation of the algorithm's performance.

6. Git Log