Runtime Comparison of Kruskal's and Prim's Minimum Spanning Tree Algorithms on Sparse and Dense Graphs

Alperen Kantarcı, Onat Şahin

Abstract

Finding the minimum spanning tree of a graph is a major computer science problem since many fields like construction, telecommunication and transportation require connecting different places with the minimum cost. The two most well-known algorithms that solve this problem are Prim's algorithm and Kruskal's algorithm. Because of the difference between the algorithms' approach to the problem, performances of these algorithms differ greatly from each other depending on the density of the input graph, which is number of edges divided by the maximum number of edges. In this project, we compare the runtimes of our implementations of Prim's and Kruskal's algorithms on a dense graph data set and a sparse graph data set, which are also generated by us. As a result of these comparisons, we found out that Prim's algorithm works way faster than Kruskal's algorithm on dense graphs. On very sparse graphs, we were expecting Kruskal's algorithm to perform better than Prim's algorithm, however Prim's algorithm performed slightly better with a very small difference between the runtimes. We expect the reason for this to be our implementation of the algorithms and graphs. The main outcome of these experiments is that as density increases, Prim's algorithm becomes more and more advantageous than Kruskal's algorithm.

Table of Contents

1. Introduction

This report is written by Alperen Kantarcı and Onat Şahin for Damien Jade Duff to present the results of the proposed project. The purpose of this project is to present a runtime comparison of the two most well-known minimum spanning tree algorithms, which are Prim's and Kruskal's algorithms, on sparse and dense graphs to get an idea on when to use which algorithm.

Finding minimum spanning trees are important for many areas and industries including computer science, telecommunication and construction. When this problem is represented with graphs, it can be seen that density, which is a relation between number of nodes and edges in a graph, is a very important factor determining the performance of the minimum spanning tree algorithms. However, during researching for the project, we have seen that there are not many research papers which compare the performances of minimum spanning tree algorithms on graphs with different densities and present the results in a simple way. The motivation for this project is to provide the reader with a comparison of Prim's and Kruskal's algorithms' performances on dense and sparse graphs and present the results in a simple way.

There are six sections in this report following introduction. In the Background section, the state of affairs are discussed, proposal of this project is summarized and information about related theoretical concepts like graphs and minimum spanning tree algorithms are given. Then, in the Methods and Design section, our implementations of these concepts are described. In the Results section, results of the experiments are presented in a graphical form, and in the Analysis section, these results are analyzed. The project's progress is outlined in the Project Evaluation section and work schedule of the project is given. Finally, the Conclusion section concludes the report by summarizing the outcomes of the project.

2. Background

2.1 The State of Affairs

Since finding the minimum spanning tree is important for many research areas, lots of studies are made regarding this topic. The first method of finding the minimum spanning tree was developed by Borůvka in 1926. After the emergence of graph theory in the 1950s, the topic of minimum spanning trees was revisited by many researchers. The two most important of these researchers are Prim and Kruskal, who developed the algorithms which are the two canonical algorithms solving the problem of minimum spanning trees. Prim's and Kruskal's algorithms are the main algorithms discussed in this project. After Prim and Kruskal, many different methods were introduced by different researchers like randomization and solving of different special cases. One of the most important of these methods was created by Graham and Hell, who used dynamic algorithms and computational models in their implementation [8].

2.2 Proposal Summary

In the proposal for this project, we discussed the two most important algorithms that solves the minimum spanning tree problem, which are Prim's and Kruskal's algorithms. We planned to compare the performances of these algorithms on dense and sparse connected graphs. A different approach could be comparing different implementations of these algorithms with each other since we only use one implementation on different datasets.

2.3 Relevant Theoretical Concepts

2.3.1 Graphs

Graph is a data structure used mainly in graph theory and discrete mathematics which represents entities in a space, some of which are associated to one another. In a graph implementation, node structures represent the entities while edges represent the associations between these entities [6].

In an undirected and connected graph:

Minimum number of edges $= V - 1$

Maximum number of edges = $\frac{V(V-1)}{2}$ (V = number of nodes)

The relation between the number of nodes and the number of edges can be represented with the density of the graph. Graphs can be classified as sparse or dense depending on their density. Examples of sparse and dense graphs can be seen ın Figure 1.

Sparse Graphs: If the number of edges of a graph is close to the minimum number of edges possible, then this graph can be classified as sparse.

Dense Graphs: If the number of edges of a graph is close to the maximum number of edges possible, then this graph can be classified as dense. In other words, "if a graph is fully connected or close to being fully connected, it can be classified as a dense graph." [6].

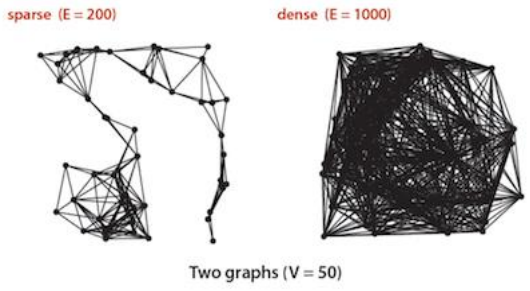

Figure 1: Sparse and dense graph examples with 50 nodes [5].

A structure which connects all nodes of a graph with minimum number of edges is called the minimum spanning tree of the graph. There can be multiple minimum spanning trees of a graph.

2.3.2 Algorithms

2.3.2.1 Prim's Algorithm

Even though this algorithm is named Prim's algorithm, it was actually created by Czech mathematician Vojtech Jarník in 1930. The algorithm was later brought back in the spotlight by Robert C. Prim in 1957 in his paper "Shortest Connection Networks and Some Generalizations". After Prim, Dijkstra also republished this algorithm in 1959. Because of all these people, this algorithm is known by many names other than Prim's algorithm, which are: JP algorithm, Jarník's algorithm, the Prim–Jarník algorithm and Prim–Dijkstra algorithm [4].

Prim's algorithm is one of the well-known and classic algorithms in the history of computer science. The algorithm's logic is very simple. It traverses graphs layer by layer. Firstly, it starts from one of the nodes of the graph and connects this node to one of its closest adjacent nodes. What we get is a tree with 2 nodes. In each step, the algorithm looks at the adjacent nodes to this tree and adds one of the closest ones to the tree. In the end, when all the nodes are visited, the tree we got is the minimum spanning tree of the graph. Steps of this algorithm are visually shown in Figure 2.

The main advantage of this algorithm is its "layer by layer" approach. In each step, the algorithm has to look at only one "layer" of the graph, which is the set of nodes adjacent to the formed tree. Because of this approach, each step requires a small amount of data no matter how big the graph is [9]. Furthermore, the algorithm requires a small number of computations and a small memory space because of this. This increases the algorithm's efficiency on dense graphs [7].

Depending on the data structure used, Prim's algorithm can be implemented with different complexities. Usage of adjacency matrices result in an $O(V^2)$ implementation, while an $O(ElogV)$ complexity can be achieved using binary heaps [8].

2.3.2.2 Kruskal's Algorithm

Joseph Kruskal is an American mathematician and computer scientist. His best-known algorithm is one of the most popular algorithms in the computer science. Kruskal's approach to finding the minimum spanning tree of a graph is so simple but powerful. The main idea of the algorithm is sorting all edges of the graph and adding these ordered edges one by one to a tree until all nodes are visited and the minimum spanning tree is achieved. Yet, there can be cycles while adding edges and by the definition of the minimum spanning tree, it cannot contain any cycles. So the algorithm should check if an edge will create a cycle if added. If it does not create a cycle, it will be added to the tree. Steps of this algorithm are visually shown in Figure 3. There are different methods for finding cycles and these methods differ for different data structures. For example, if the disjoint-set data structure is used, then the union-find algorithm can be used for cycle detection. Different data structures and different sorting methods will change the complexity of the algorithm. The first operation is sorting the edges which takes $O(ElogE)$. The only other non-trivial operation is cycle detection. If we consider that disjoint-set

data structure is used then all cycle detections will take $O\big(E\alpha(E,V)\big)$. As Mareš stated, other than the disjoint-set data structure, simpler data structures can also be used as long as amortized cost of each operation is $O(logV)$ since the complexity of the sorting will dominate the cycle detection and

general algorithm's runtime wouldn't be affected [8]. Further studies on Kruskal's algorithm shows that it can be faster with the dynamic maintenance of connected components.

As density increases in the graph, number of edges of the graph also increases, so the sorting part will take longer to finish. The cycle detection time will increase as well. Therefore, its performance will drop as density increases while number of nodes stays same.

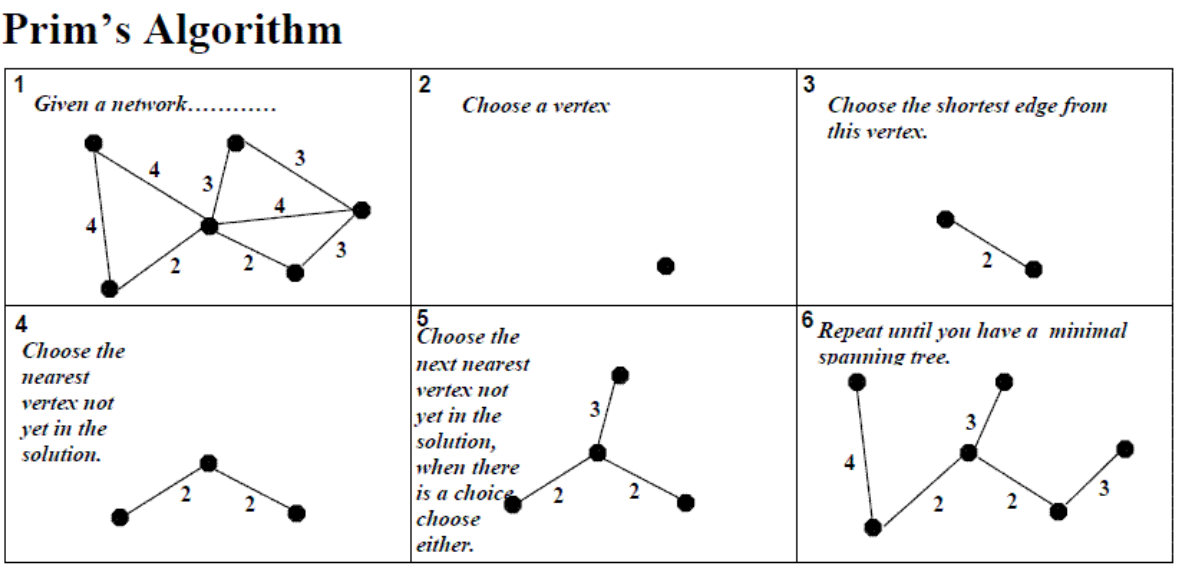

Figure 2: Steps of Prim's Algorithm [2]

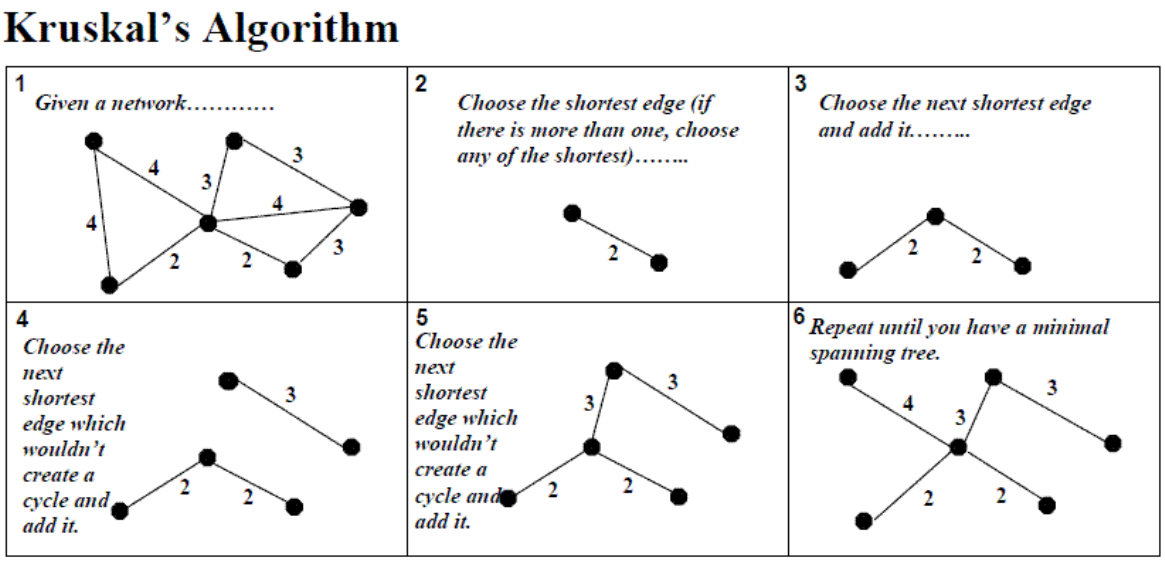

Figure 3: Steps of Kruskal's Algorithm [2]

3. Methods & Design

For this project, we made two experiments. The first one is comparing Kruskal's and Prim's algorithms on sparse graphs and the second one is comparing these algorithms on dense graphs. We made the first experiment to see which algorithm performs better on sparse graphs. Likewise, we made the second experiment to find out which algorithm performs better on dense graphs. For these experiments, we first needed to create datasets that include sparse and dense graphs. Then, we implemented Kruskal's and Prim's algorithms. We used a computer with an Intel Core i7 4700MQ processor. We made our implementations in Python 3.6.4.

3.1. Graph and Data Set Generation

For input data, we created our own random-connected graph generation algorithm. This algorithm takes the number of nodes and the density of the graph as inputs. Here, we define density as number of edges divided by the maximum possible number of edges for the given number of nodes. We represent our graphs with adjacency matrices. In each step, our algorithm connects a new node to another randomly chosen node (makes an element of the adjacency matrix 1). After this is complete, we get a graph with V nodes and V-1 edges. Then, our algorithm proceeds to add randomly selected edges to the graph until the desired density is achieved. Also, whenever an edge is added to the graph, we assign a random weight to that edge, ranging from 1 to 100. As a result, the algorithm returns a randomly generated, connected and weighted graph with given number of nodes and density. Example graphs with densities 0.02 and 0.9 generated using this algorithm can be seen in Figure 4 and Figure 5, respectively.

Using the given algorithm, we created two data sets, one of which is sparse and one of which is dense. Each data set contains 190 graphs with 100 nodes. In the sparse graph data set, density ranges from 0.02 to 0.20. We acquired different densities with the step size 0.01 in this range. This means the dataset contains 19 different densities. For each density, we generate 10 graphs to see the average, minimum and maximum results. This enables us get a more accurate result. In the dense graph data set, everything is the same except the density range, which is from 0.72 to 0.9.

3.2 Kruskal's Algorithm

In our implementation of Kruskal's algorithm, we first sort the edges of the graph in terms of their weights in $O(ElogE)$. Then, starting from the minimum weighted edge, we add the edge to the minimum spanning tree if the edge does not create a cycle until all nodes are covered by the minimum spanning tree.

The important part of this algorithm is the cycle detection. For this part, we used an approach that uses the depth first search algorithm. Whenever the algorithm tries to decide whether an edge will create a
cycle, the edge is added to the minimum spanning tree and depth first search is run. If the search revisits an already visited node, this means the tree contains a cycle. In this case, the added edge is removed. Otherwise, the edge is kept. Since we use adjacency matrices for our implementation, this

step has a complexity of $O(V^2)$. Except in very sparse graphs, number of edges (E) is much bigger than number of nodes (V). For this reason, the part where we sort the edges generally takes the most time when the algorithm is run.

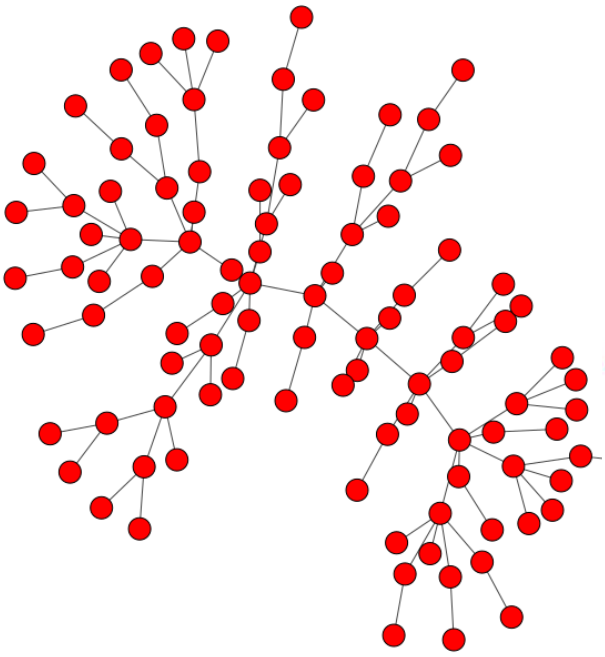

Figure 4: Sparse graph with a density of 0.02 and 100 nodes

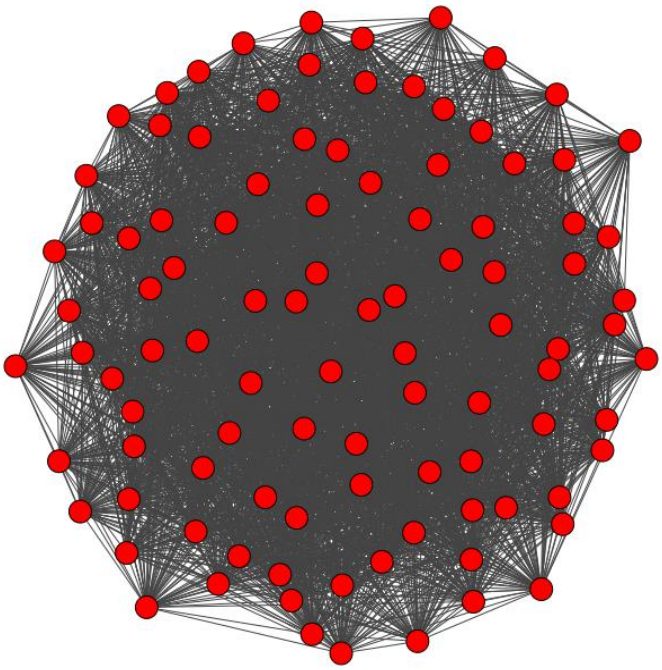

Figure 5: Dense graph with a density of 0.9 and 100 nodes

3.3 Prim's Algorithm

In our implementation of Prim's algorithm, we first create 3 arrays. The first array is named mstSet and contains the nodes which are decided to be in the minimum spanning tree. This array is initialized with one of the nodes. The second array is named mstNotSet and contains the nodes which are not in the minimum spanning tree. This array is initialized with indexes of every node in the graph except the one in the mstSet. The final array is called mst, and this array contains the edges in the minimum spanning tree and initialized empty. In each step, for each node in the mstSet, we traverse mstNotSet and look for nodes which are connected to the current node in the graph. We calculate the minimum weighted edge which connects the current node to a connected node in mstNotSet and add this edge to the mst. Finally, we remove the node which is in the mstNotSet and add it to the mstSet. We repeat this until mstNotSet is empty. After this is complete, mst will contain the edges of the minimum spanning tree of the graph. Since we traverse every adjacent node for every node in the graph, the complexity of this algorithm is $O(V^2)$.

4. Results

We made two experiments. The first experiment is comparing the runtimes of our implementations of Prim's and Kruskal's algorithms on our sparse graph data set. The results of this experiment can be seen in Figure 6. The second experiment is comparing the runtimes of our implementations of Prim's and Kruskal's algorithms on our dense graph data set. The results of this experiment can be seen in Figure 7.

In these plots, x-axis represents density of the graph, while y-axis represents time in seconds. Since we tested our algorithms on ten different randomly generated graphs of the same density for each density, data points in these plots represent the average runtime of these ten tests. Maximum and minimum runtimes of these tests are represented with the tips of the vertical lines that pass through each data point.

5. Analysis

Our first experiment was to compare the runtimes of Prim's and Kruskal's algorithms that we implemented on sparse graphs. Plot of the results of this experiment can be seen in Figure 6. From this plot, it can be seen that on very sparse graphs, Kruskal's algorithm's performance is really high. In the background section, we mentioned that Kruskal's algorithm is expected to work faster than Prim's algorithm on very sparse graphs. However, in the results we get, Prim's algorithm works slightly faster than Kruskal's algorithm on even very sparse graphs. This could be a result of our implementation of graphs and algorithms. We used adjacency matrices for our graph structure and depth first search for cycle detection. In the background section, it is mentioned that the runtime of Kruskal's algorithm can be improved by using list data structures for graphs and union-find algorithm that uses disjoint-sets for cycle detection. We expect this to be the main problem of our implementation.

Our second experiment was to compare the runtimes of Prim's and Kruskal's algorithms that we implemented on dense graphs. Plot of the results of this experiment can be seen in Figure 7. From this plot, it can be seen that Prim works faster than Kruskal's algorithm on dense graphs. This outcome does not contradict the expectations stated in the background section. Though, the possible problems of the implementation of Kruskal's algorithm makes the gap between the runtimes of these algorithms seem larger than it should be. Still, this plot correctly shows that it is more advantageous to use Prim's algorithm on dense graphs.

Even though the results gathered are not entirely correct, they still show the main expected result: as density increases, Prim's algorithm becomes more and more advantageous than Kruskal's algorithm. Our sparse graph data set includes graphs with densities ranging from 0.02 to 0.20 and our dense graph data set includes graphs with densities ranging from 0.72 to 0.90. In both experiments, as density increases, the gap between the runtimes of Prim's and Kruskal's algorithm widens.

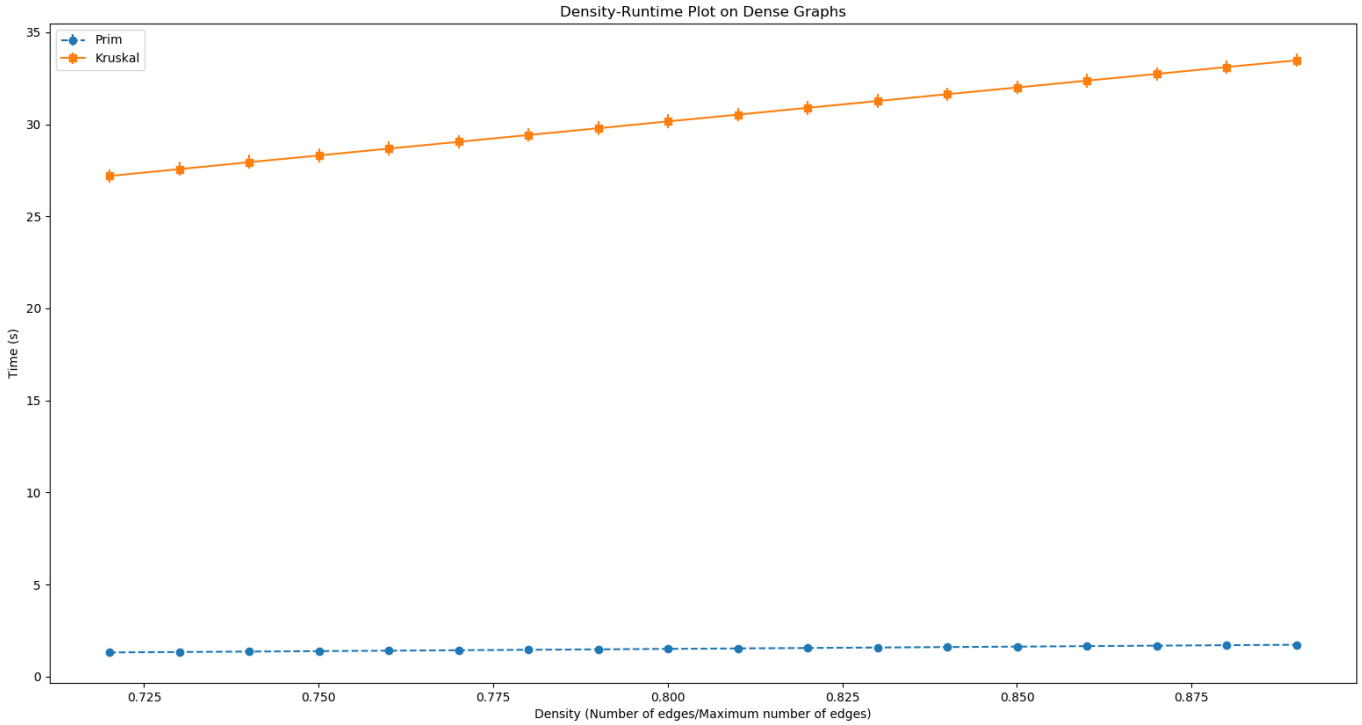

Figure 6: Experiment run time results on the sparse dataset

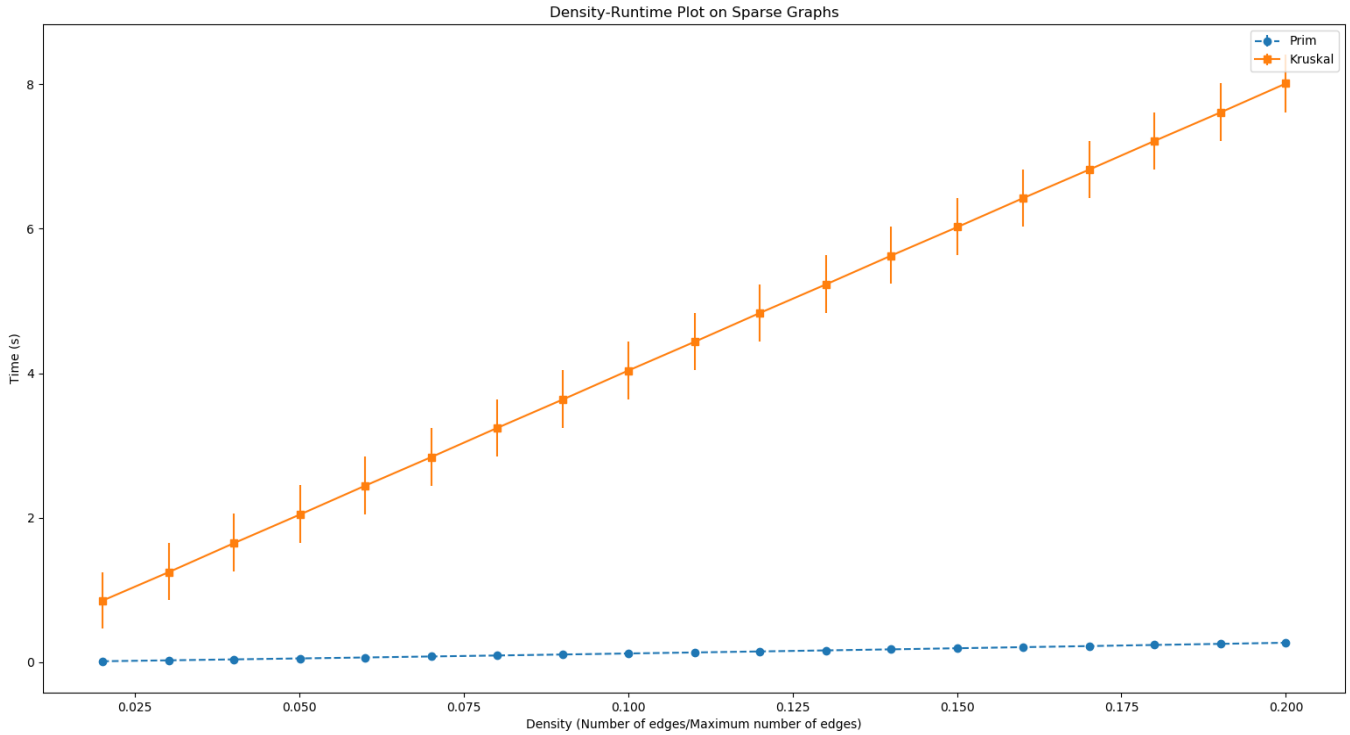

Figure 7: Experiment run time results on the dense dataset

6. Project Evaluation

6.1 Project Process Summary

Firstly, we looked for graph data sets to work on. We found some, but none of them were really suited for our purpose. Then, we found the python module called igraph [1]. This module seemed like what we were looking for, because it included functions which generate random graphs with a given node number and density. However, these graphs were not connected. Since we were going to find minimum spanning trees, we needed connected graphs. Because of this, we decided to implement our own graph structure and graph generation algorithm. Then, we implemented Prim's and Kruskal's algorithm. Implementing Kruskal's algorithm was especially tricky and caused us problems. Then, we made our experiments and gathered results. Finally, we used results to create Figure 4 and Figure 5 using matplotlib module of python [3]. We also visualized one example graph from each data set to create Figure 2 and Figure 3 using the plotting function of igraph.

6.2 Schedule

Figure 8 shows our initial schedule as a Gantt chart. The table in Figure 9 shows the finish dates of the tasks.

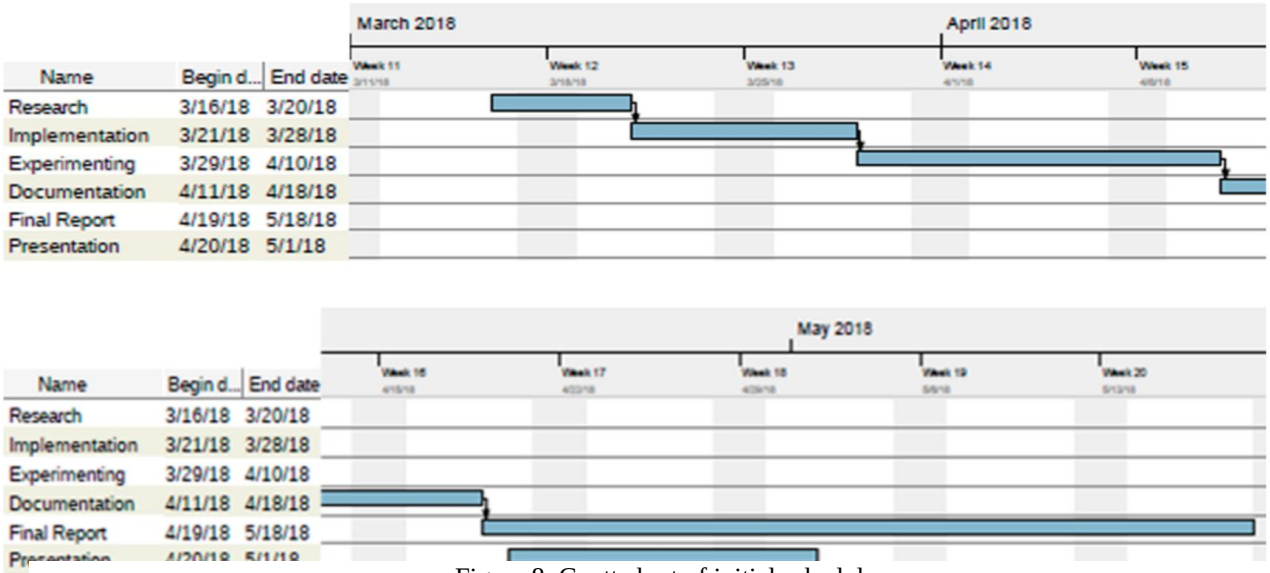

Figure 8: Gantt chart of initial schedule

Tasks	Original Due Date	Finish Date
Research	20/03/2018	20.03.2018
Implementation	28/03/2018	12/04/2018
Experimentation	10/04/2018	19/04/2018
Documentation	18/04/2018	22/04/2018
Final Report	25/04/2018	02/05/2018
Presentation	01/05/2018	Expected to finish before 18/05/2018

Figure 9: Due and finish dates of project tasks

7. Conclusion

To conclude, we compared the runtimes of our implementations of Kruskal's and Prim's algorithms on our sparse and dense graph data sets. We represented the results of these experiments in a graphical form. As a result, we found that as density increases, performance of Prim's algorithm stays mostly the same while performance of Kruskal's algorithm gradually gets worse. We were expecting Kruskal's algorithm to perform better that Prim's algorithm on very sparse graphs, however we were unable to observe this. We think this is a result of our implementation of the algorithms and our graph structure. We also concluded that Prim's algorithm is much easier to implement than Kruskal's algorithm. Given it's performance on dense graph, it can be said that Prim's algorithm should be the preferred minimum spanning tree algorithm unless the graphs being worked on are very sparse.

8. References

[1] Anon. igraph – The network analysis package. Retrieved April 27, 2018 from http://igraph.org/

[2] Anon. Kruskal vs Prim. Retrieved May 1, 2018 from https://stackoverflow.com/questions/1195872/kruskal-vs-prim

[3] Anon. Matplotlib: Python plotting - Matplotlib 2.2.2 Retrieved April 27, 2018 from https://matplotlib.org/

[4] Anon. 2018. Prim's algorithm. (April 2018). Retrieved April 27, 2018 from https://en.wikipedia.org/wiki/Prim's_algorithm

[5] Anon. Representing Graphs. Retrieved May 1, 2018 from http://www.oxford-mathcenter.com/drupal7/node/646

[6] Kantarci A. and Şahin O. 2018. Comparison of Computational Performance of Minimum Spanning Tree Algorithms 1-2

[7] A. Kershenbaum and R.Van Slyke. 1972. Computing minimum spanning trees efficiently. Proceedings of the ACM annual conference on - ACM72 (1972). DOI: http://dx.doi.org/10.1145/800193.569966

[8] Martin Mareš. 2008. The saga of minimum spanning trees. Computer Science Review 2, 3 (2008), 165–221. DOI: http://dx.doi.org/10.1016/j.cosrev.2008.10.002

[9] R.C. Prim. 1957. Shortest Connection Networks And Some Generalizations. Bell System Technical Journal 36, 6 (1957), 1389–1401. DOI:http://dx.doi.org/10.1002/j.1538-7305.1957.tb01515.x