Program 2 report

Amey Meher (avmeher), Deep Mehta (dmmehta2) and Gage Fringer (gwfringe)

April 12, 2023

Abstract

This report covers the experimental analysis of three graph algorithms, namely Prim Jarnik, Krushkal, and Boruvka. The report first gives an overview of the graph algorithm, then discusses the theoretical analysis of the three algorithms. It then proceeds to describe the experimental setup, along with the specifics of the implementation. Different types of analyses are conducted, namely runtime analysis of these experiments, comparisons of the number of comparisons by varying the number of vertices and edges, as well as a few others discussed in the results section. The report concludes the fact that the graph algorithm's efficiencies vary based on numerous factors and each algorithm performs better than the others in certain situations.

1 Introduction

A graph is a structure that consists of a set of vertices and a set of edges that connect pairs of vertices. The edges may be directed or undirected.

A minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

There are several algorithms for finding minimum spanning trees. The ones which we will focus on in this paper, are Boruvka's, Krushkals and Prim Jarnik.

2 Theory

Table 1 gives the theoretical run-time analysis of the three algorithms. We have considered the runtime analysis of Prim Jarnik based on binary heap implementation because that is what we have utilized in Program2.

We also aim to identify which algorithm performs better in what scenario because they have the same theoretical time complexity. Although it heavily depends upon the implementation of the algorithms, we have specified a few parameters for all algorithms which would help in comparing the algorithms. This will be discussed in further detail in the experimental analysis section.

Algorithm	**Runtime**
Boruvka	$\mathcal{O}(mlgn)$
Krushkal	$\mathcal{O}(mlgn)$
Prim Jarnik (Binary Heap)	$\mathcal{O}(mlgn)$

Table 1: Theoretical Run-time analysis

3 Experimental design

Below are a few of the things that we have kept in common for all the implementations which would ensure the algorithms are compared in a fair manner.

1. The input is in the form of a .gph file, which contains a count of vertices and edges, as well as information specific to each node and edge in a list.
2. The total time taken does not take into consideration the time to read the input from the user or other auxiliary tasks such as printing the output. It only considers the time to sort the list.
3. Each value provided comes as an average of five trials performed on a specific input file and algorithm combination to ensure that outlier executions are minimized.
4. All calculations done on where runtime was the main measurement were done on the following hardware:
   - Processor: Intel Core i7-8565U @ 1.80 GHz
   - RAM: 16GB (2400MHz)
   - L1 Cache: 256 KB
   - L2 Cache: 1.0 MB
   - L3 Cache: 8.0 MB

Below is the set of experiments we thought of conducting with the three algorithms:

1. **Three-way comparison of Boruvka's Kruskal's and Prim's Algorithms:** Running each of these algorithms on a variety of inputs to determine how their behaviors compare relative to each other, as well as against their expected runtime.
2. **Vertices vs Comparisons:** Analyzing how the number of comparison varies with number of vertices across all three algorithms and across various types of graphs.
3. **Edges vs Comparisons:** Analyzing how the number of comparison varies with number of edges across all three algorithms and across various types of graphs.

Assumptions made specific to the three-way comparison experiment are as follows:

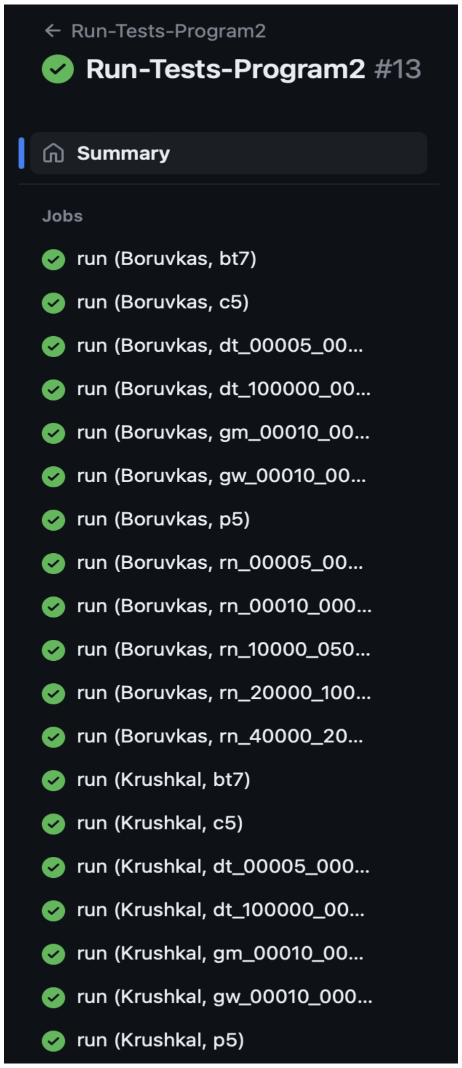
1. All input files for this experiment are unmodified copies of those provided in the "ProgramTwo-Student-Files" folder. The applicable files are all files present except for those listed below, as the time taken to read input was not reasonable for getting multiple trials to confirm averages.
   - gm_100000_3000000_100000_01.gph
   - gw_100000_3000000_100000_01.gph
   - rn_1000000_3000000_100000_01.gph

4 Implementation

We have implemented all three graph algorithms as Java classes. The input data is then passed to these algorithms through a shell script wrapper. Also, for generating input graphs, we have made use of the generate.sh shell script.

Correctness test:

We tested all the graph algorithms on all the input files provided in an automated workflow. Below is the screenshot of a run of the workflow which confirms that the output from all the run is as expected and matching the expected output.

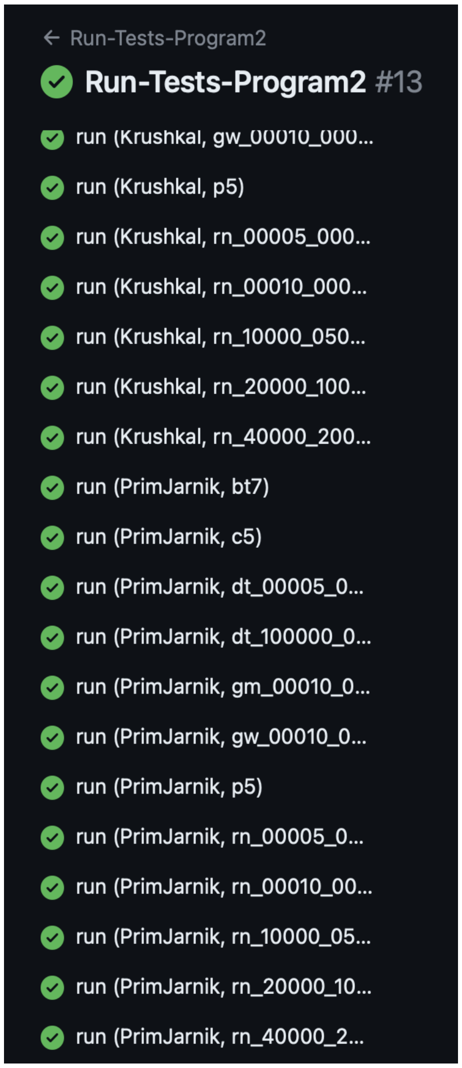


```
← Run-Tests-Program2
Run-Tests-Program2 #13

Summary

Jobs
run (Boruvkas, bt7)
run (Boruvkas, c5)
run (Boruvkas, dt_00005_00...
run (Boruvkas, dt_100000_00...
run (Boruvkas, gm_00010_00...
run (Boruvkas, gw_00010_00...
run (Boruvkas, p5)
run (Boruvkas, rn_00005_00...
run (Boruvkas, rn_00010_000...
run (Boruvkas, rn_10000_050...
run (Boruvkas, rn_20000_100...
run (Boruvkas, rn_40000_20...
run (Krushkal, bt7)
run (Krushkal, c5)
run (Krushkal, dt_00005_000...
run (Krushkal, dt_100000_00...
run (Krushkal, gm_00010_00...
run (Krushkal, gw_00010_000...
run (Krushkal, p5)
```

Part 1



```
← Run-Tests-Program2
Run-Tests-Program2 #13
run (Krushkal, gw_00010_000...
run (Krushkal, p5)
run (Krushkal, rn_00005_000...
run (Krushkal, rn_00010_000...
run (Krushkal, rn_10000_050...
run (Krushkal, rn_20000_100...
run (Krushkal, rn_40000_200...
run (PrimJarnik, bt7)
run (PrimJarnik, c5)
run (PrimJarnik, dt_00005_0...
run (PrimJarnik, dt_100000_0...
run (PrimJarnik, gm_00010_0...
run (PrimJarnik, gw_00010_0...
run (PrimJarnik, p5)
run (PrimJarnik, rn_00005_0...
run (PrimJarnik, rn_00010_00...
run (PrimJarnik, rn_10000_05...
run (PrimJarnik, rn_20000_10...
run (PrimJarnik, rn_40000_2...
```

Part 2

Figure 1: Correctness test

5 Analysis

1. **Three-way comparison:**

   After all three algorithms had completed their required trials on their intended datasets, the averages were gathered such that tables could be constructed and visuals created to better understand the gathered results, as seen in the table below.

Input Size (m lg n)	Kruskal's	Boruvka's	Prim's
1.16 E01	0.00	0.00	0.00
1.29 E01	0.00	0.00	0.00
1.63 E01	0.00	0.00	0.00
1.68 E01	0.00	0.00	0.00
2.32 E01	0.00	0.00	0.00
4.98 E01	0.00	0.00	0.00
4.98 E01	0.00	0.00	0.00
6.64 E01	0.00	0.00	0.00
6.64 E05	0.11	0.65	0.14
1.43 E06	0.18	0.69	0.26
3.06 E06	0.34	1.57	0.53
4.98 E06	0.48	0.39	0.49

Table 2: Runtime analysis on provided files

   In the above table, the order of input files is as follows:

   - c5.gph (1.16 E01)
   - p5.gph (1.29 E01)
   - dt_00005_000000_0100_01.gph (1.63 E01)
   - bt7.gph (1.68 E01)
   - rn_00005_000010_0200_01.gph (2.32 E01)
   - gm_000010_000020_0500_01.gph (4.98 E01)
   - gw_000010_000020_1000_01.gph (4.98 E01)
   - rn_00010_000020_0500_01.gph (6.64 E01)
   - rn_10000_050000_50000_01.gph (6.64 E05)
   - rn_20000_100000_50000_01.gph (1.43 E06)
   - rn_40000_200000_50000_01.gph (3.06 E06)
   - dt_100000_000000_100000_01.gph (4.98 E06)

   From this table, we can construct visuals to better depict whether or not the algorithms follow their expected runtime of O(m log n). The inputs were standardized to an m log n value to be able to take into consideration both the number of vertices and edges in a specific input. For the sake of clarity, inputs small enough that the runtime is measured at 0 have been removed from subsequent visuals, as they do not contribute valuable insights.

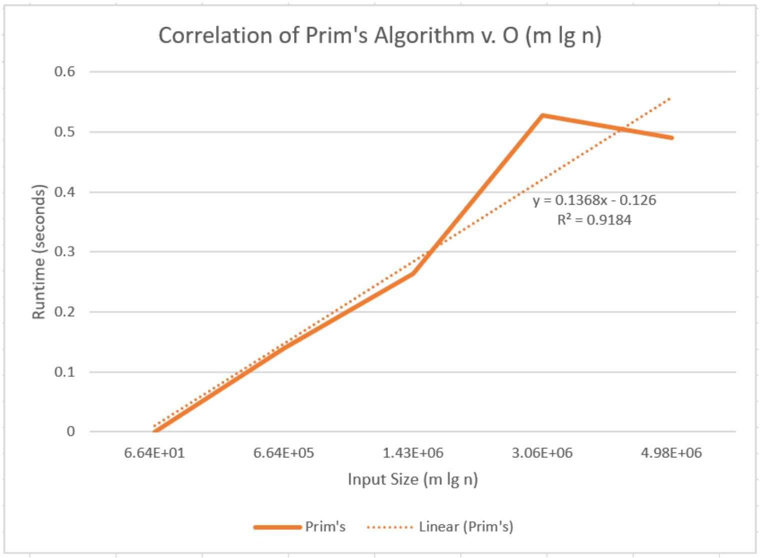


Figure 2: Prim's algorithm v. expected runtime

Looking at Prim's algorithm and the provided correlation statistics in Figure 2, we can see that regardless of the type of graph provided, Prim's algorithm seems to fairly closely follow a growth in runtime that is at the same rate as its expected runtime, with a slight variance in denser input cases, like that which is found at 3.06 E06 (rn_40000).

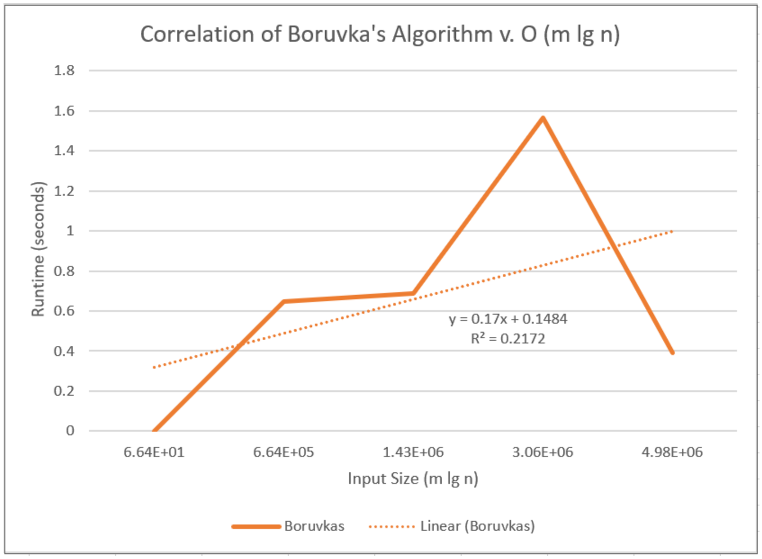


Figure 3: Boruvka's algorithm v. expected runtime

As seen in the results of Figure 3, Boruvka's algorithm seemed to have the most variance in its timing, and also fit its expected growth rate the poorest out of these three algorithm implementations. We can make the assumption that denser graphs are more costly with this algorithm, as the graph from rn_40000 is denser (5) than dt_100000 (2.99), which may explain the difference in timing.

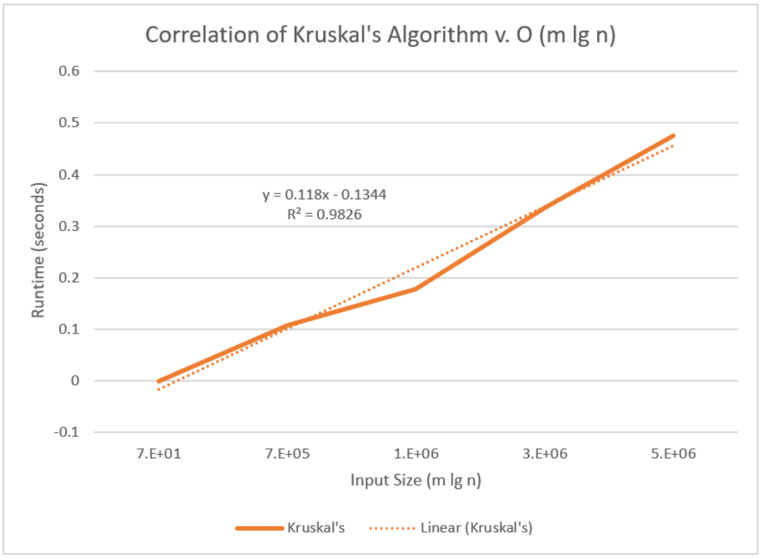


Figure 4: Kruskal's algorithm v. expected runtime

Comparing the results of Figure 4 and the trendlines of the other algorithms, we can easily conclude that Kruskal's algorithm seemed to perform the best with this variety of inputs, as we see that no particular input looked to be a very strong outlier, as was more the case with the other two algorithms with larger, denser inputs.

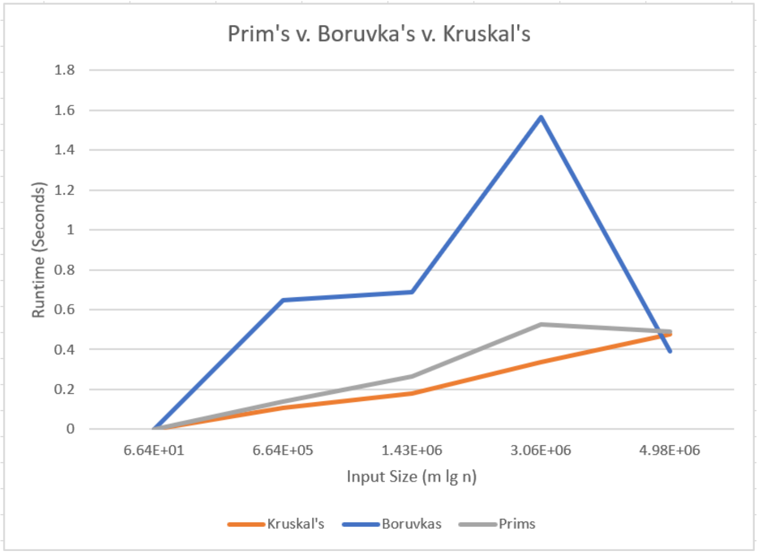


Figure 5: Runtimes of all three algorithms

As a final comparison, we can look at the relative runtimes of each algorithm against each other with the same input graphs, as seen in Figure 5. Looking at these results alongside each other, it was interesting to see that Kruskal's and Prim's algorithms had similar runtimes with the input, while Boruvka's algorithm had a larger average runtime on most inputs. This may be attributed to the union method found in the Boruvka's code, as it iterates through all vertices in a connected component, which could prove to be costly the larger the components become. This issue may be an issue specific to some of these inputs, as we can see that the file with the largest number of inputs (dt_100000) did not have the highest runtime. Though this is the case, you can see that all three algorithms follow the same general upward trend in their runtimes, further suggesting that all three of these algorithms run at their expected runtime based on the increase in input.

2. **Vertices vs Comparisons:**

   This experiment tested the input graphs generated by varying number of vertices from 50 to 500, with an increment of 5, keeping edges constant, i.e. maximum number of edges that can exist. Such input graphs were generated for each type of graph, i.e. rn, gm, gw and dt.

   These graphs were then given as inputs to all the three algorihtms, Prims, Krushkals and Boruvkas and various graphs were generated based on the output.

   (a) Boruvka's Algorithm - For all type of graphs, showing vertices vs comparisons

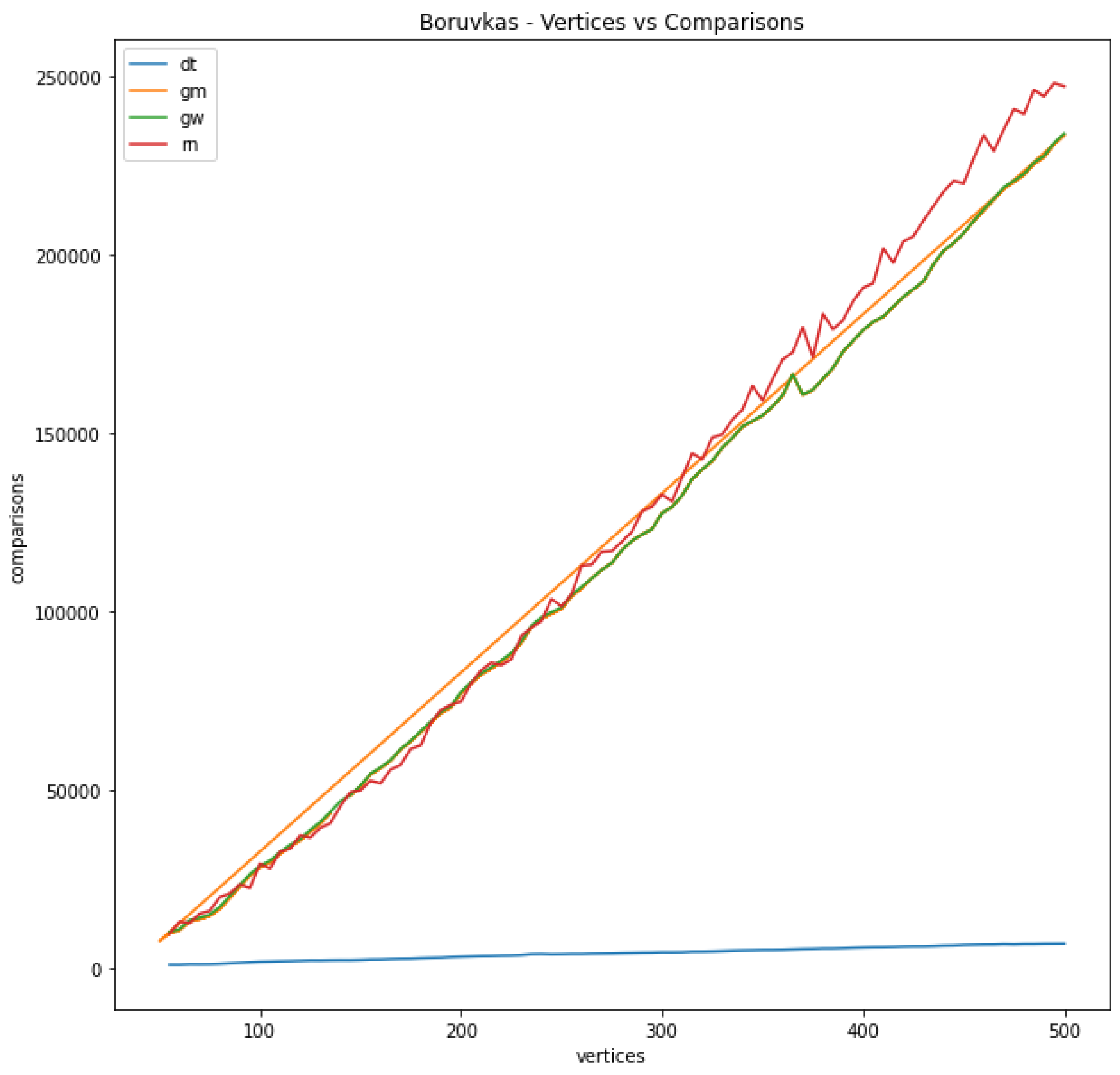


Figure 6: Boruvka's algorithm - vertices vs comparisons

We can see that for dt, Boruvka's show almost constant number of comparisons. After that the most increase in number of comparisons is seen for random graphs. Additionally we see rn, gm, gw have a linear increase in number of comparisons.

(b) Krushkal's Algorithm - For all type of graphs, showing vertices vs comparisons

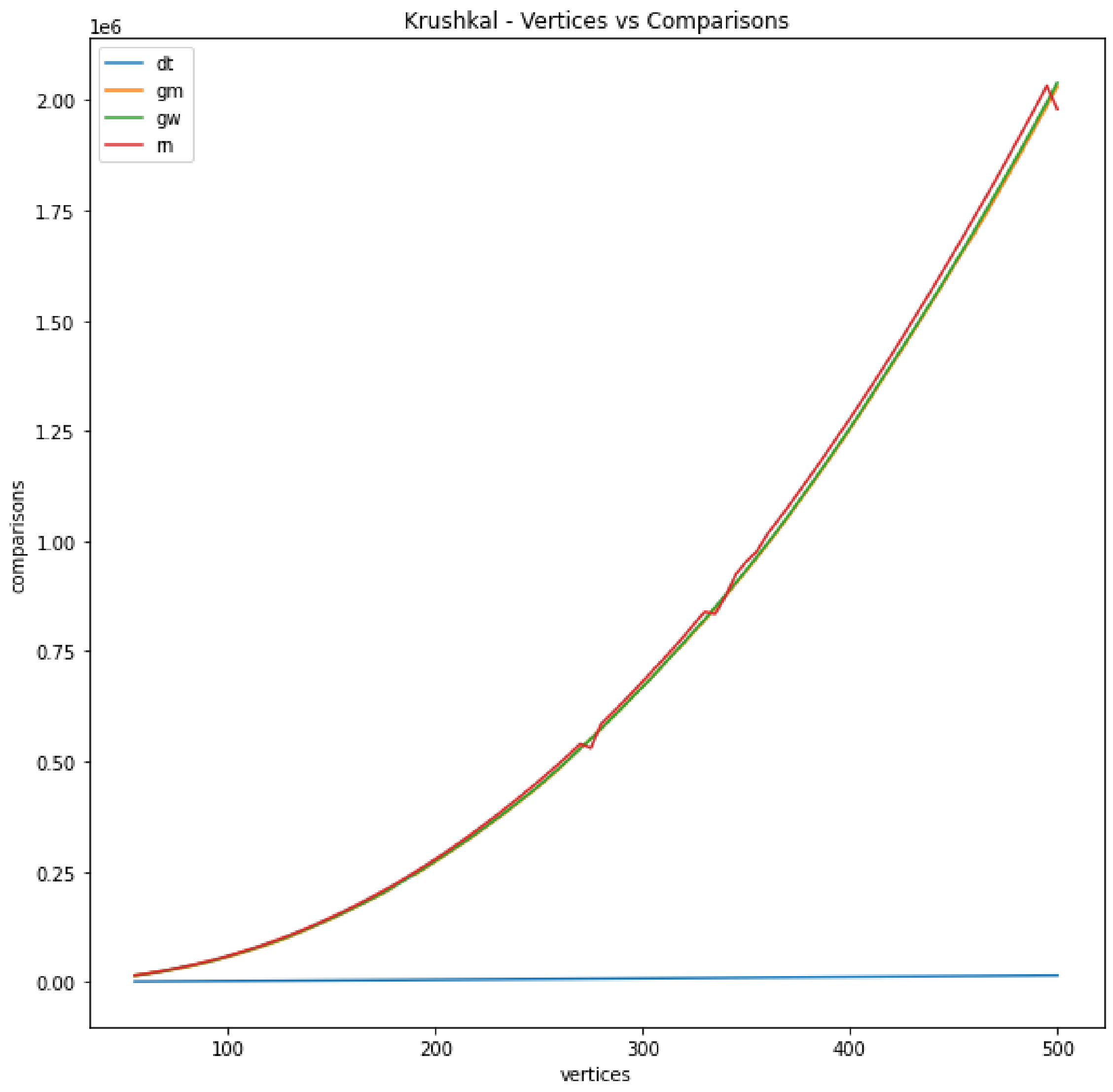


Figure 7: Krushkal's algorithm - vertices vs comparisons

We can see that for dt, Krushkal's show almost constant number of comparisons. Apart from that rn, gm and gw show a exponential increase in number of comparisons with the increase in number of vertices.

(c) Prim's Algorithm - showing vertices vs comparisons

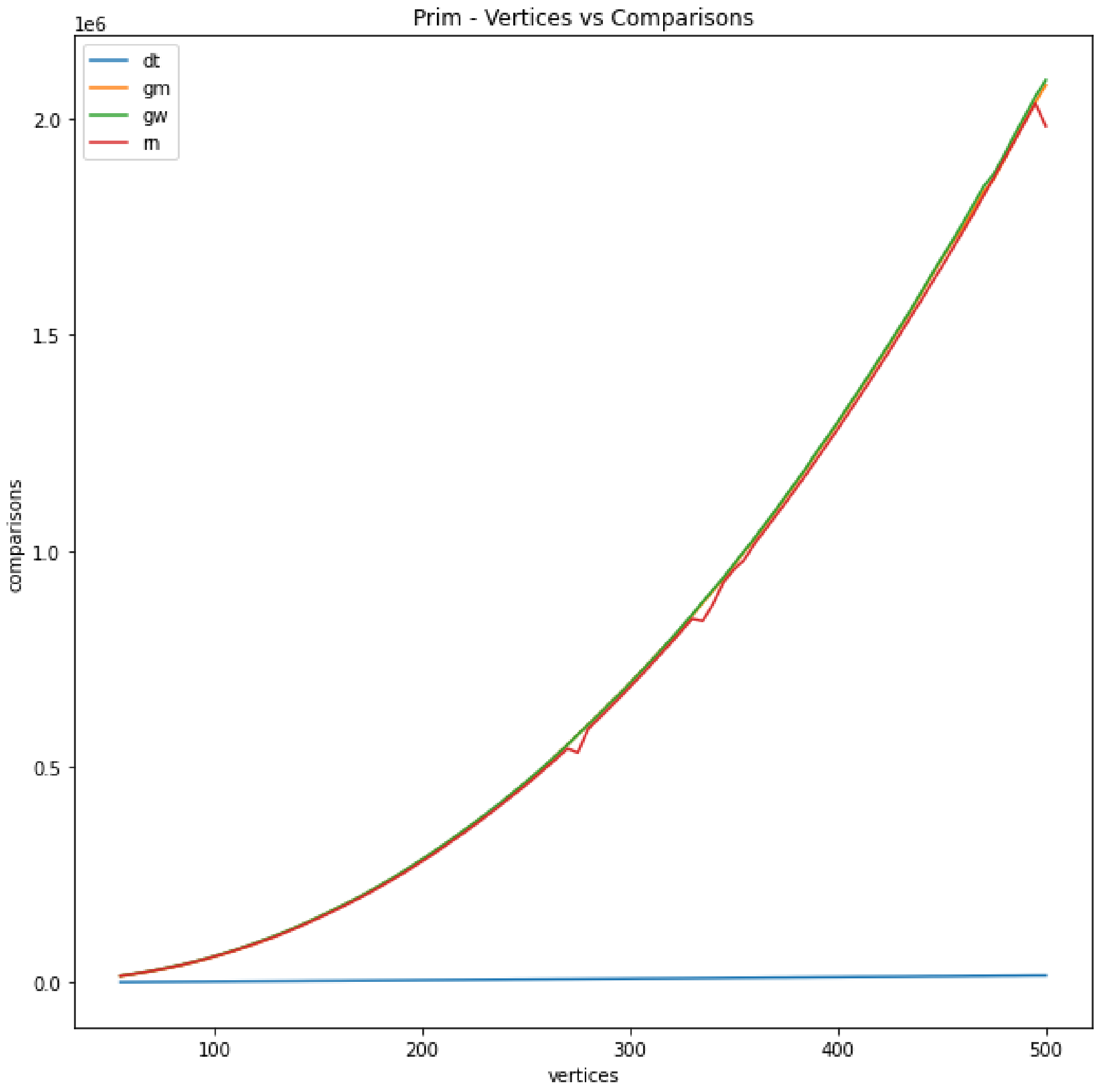


Figure 8: Prim's algorithms - vertices vs comparisons

We can see that for dt, Prim's show almost constant number of comparisons. Apart from that rn, gm and gw show a exponential increase in number of comparisons with the increase in number of vertices.

(d) All Algorithm - dt graph, showing vertices vs comparisons

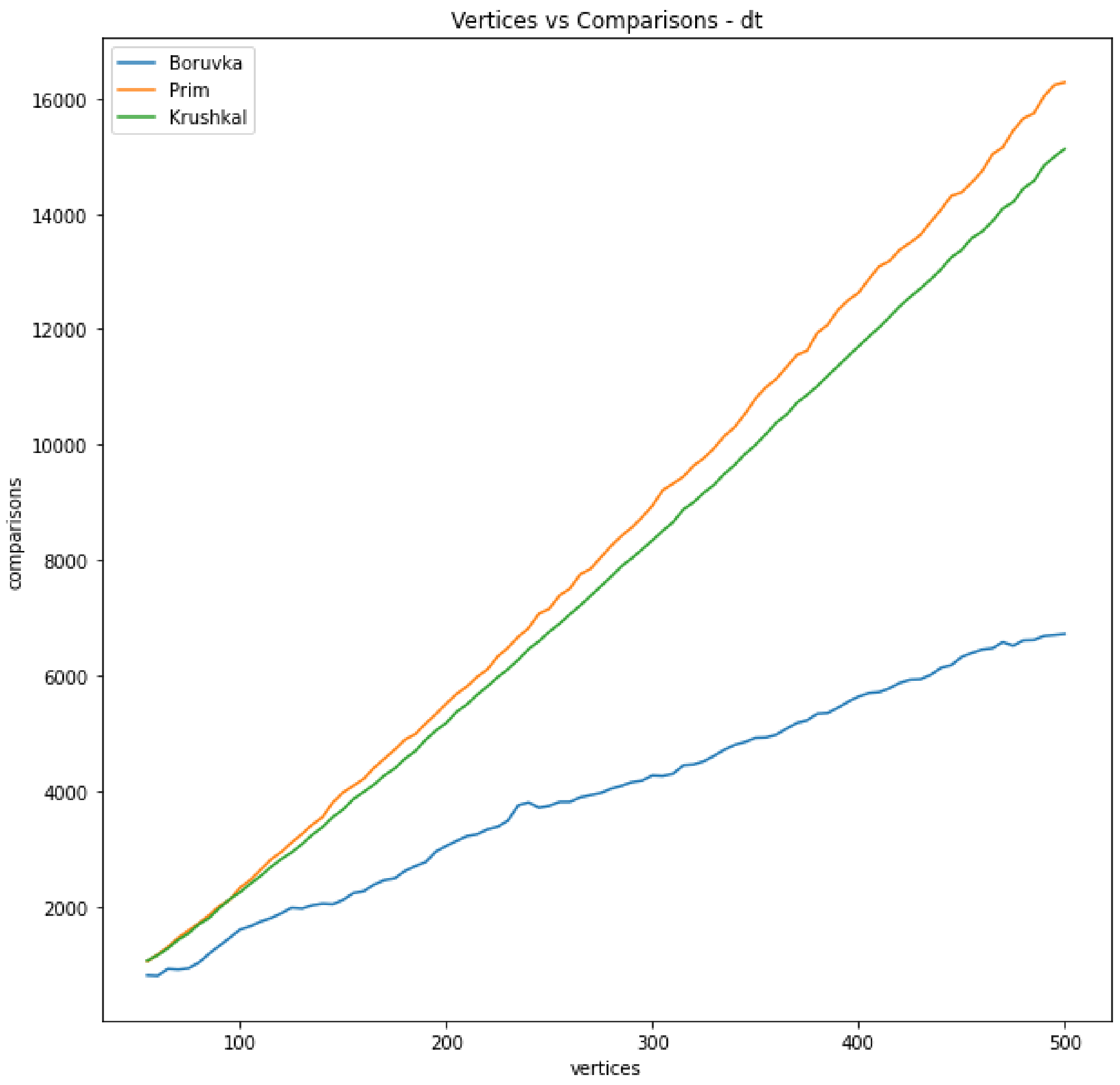


Figure 9: All algorithms - dt graph - vertices vs comparisons

We can see that Boruvka's performs much better compared to other two algorithms when we use dt graphs. Prim's and Krushkal's are pretty close in the competition, but towards the end, Prim's overtakes Krushkals and has the most number of comparisons across all three algorithms.

(e) All Algorithm - gm graph, showing vertices vs comparisons

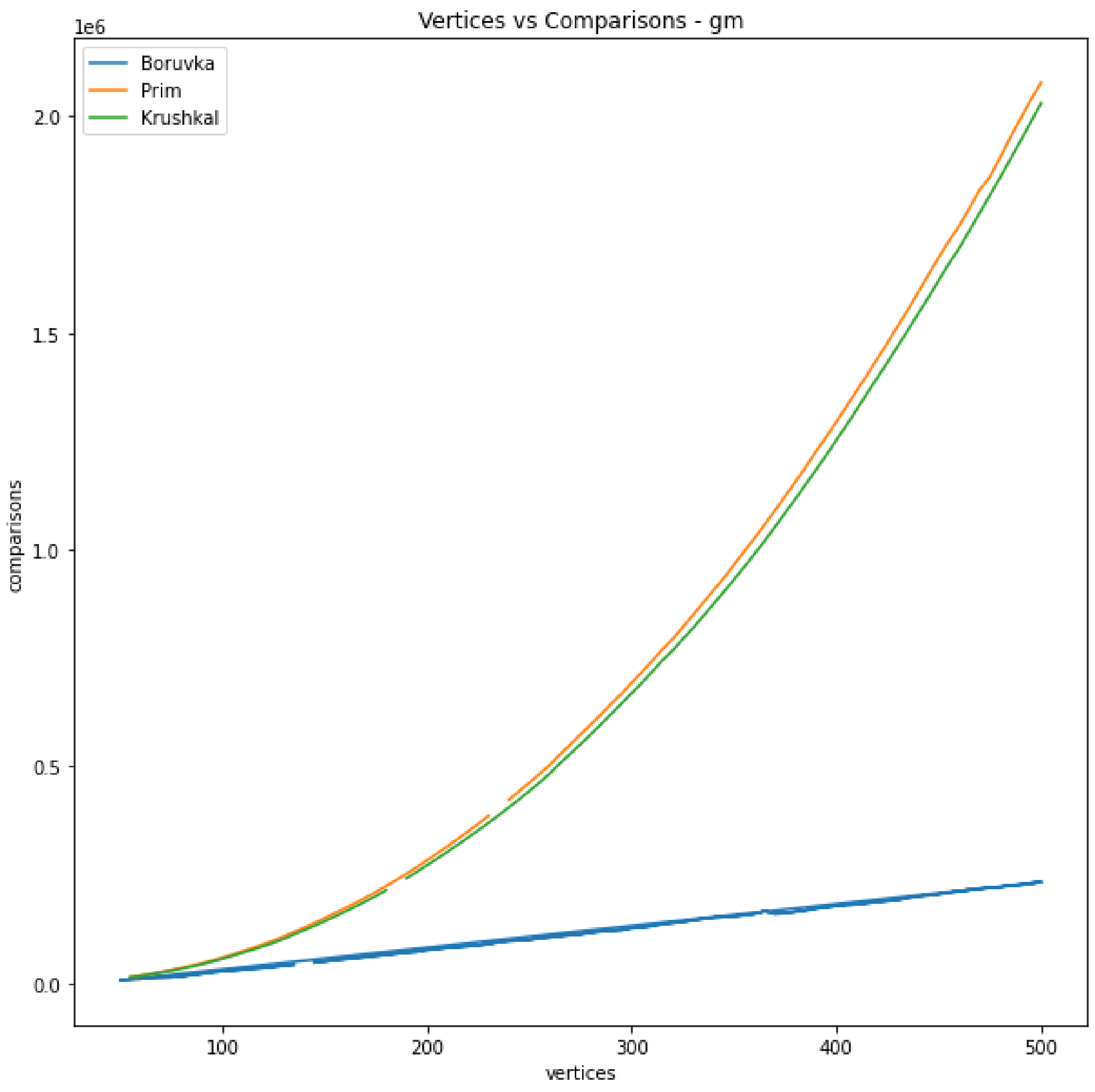


Figure 10: All algorithms - gm graph - vertices vs comparisons

We can see that Boruvka's performs much better compared to other two algorithms when we use gm graphs. Prim's and Krushkal's pretty much have the same number of comparisons across the graph.

(f) All Algorithm - gw graph, showing vertices vs comparisons

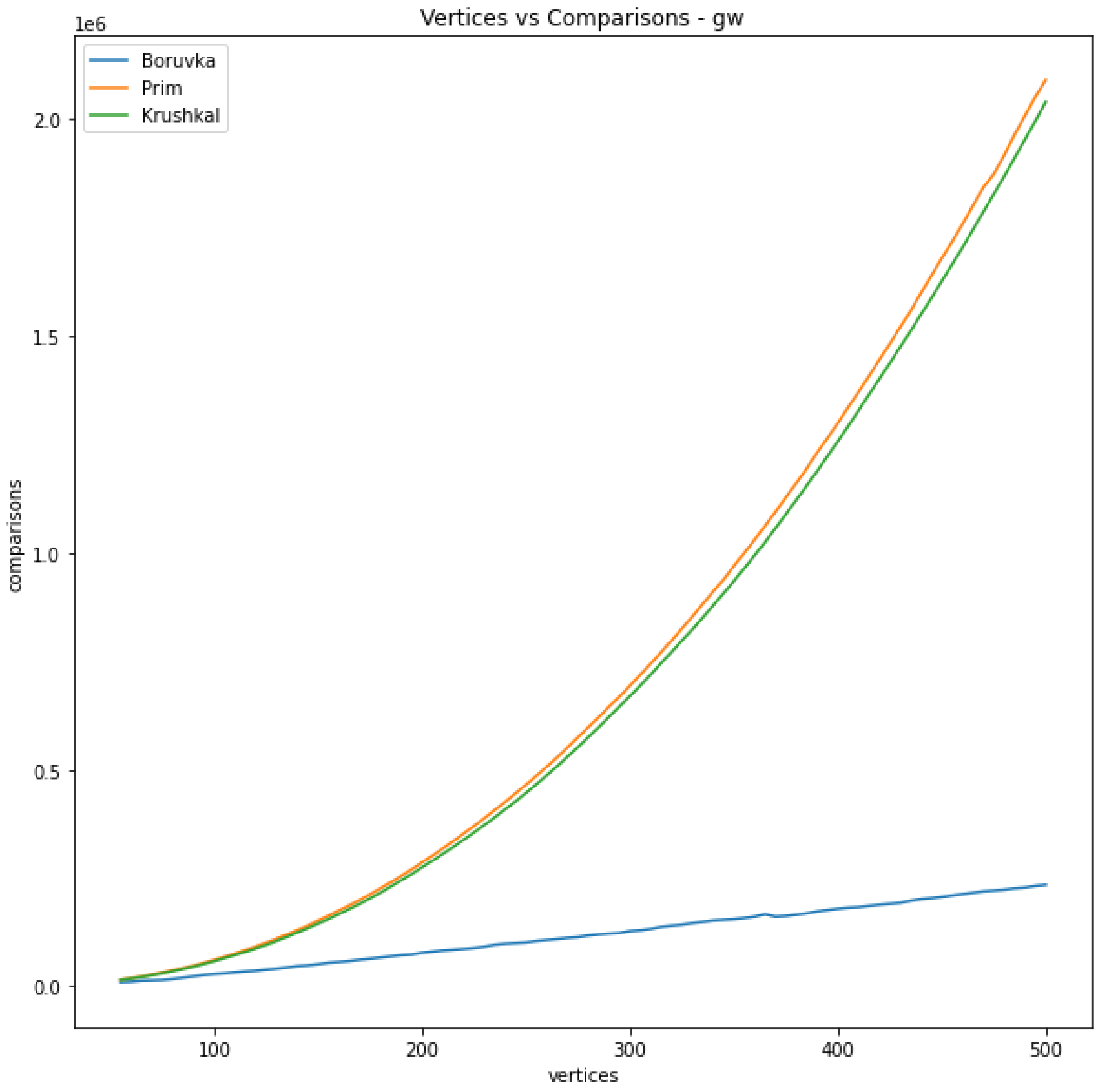


Figure 11: All algorithms - gw graph - vertices vs comparisons

We can see that Boruvka's performs much better compared to other two algorithms when we use gw graphs. Prim's and Krushkal's pretty much have the same number of comparisons across the graph.

(g) All Algorithm - rn graph, showing vertices vs comparisons

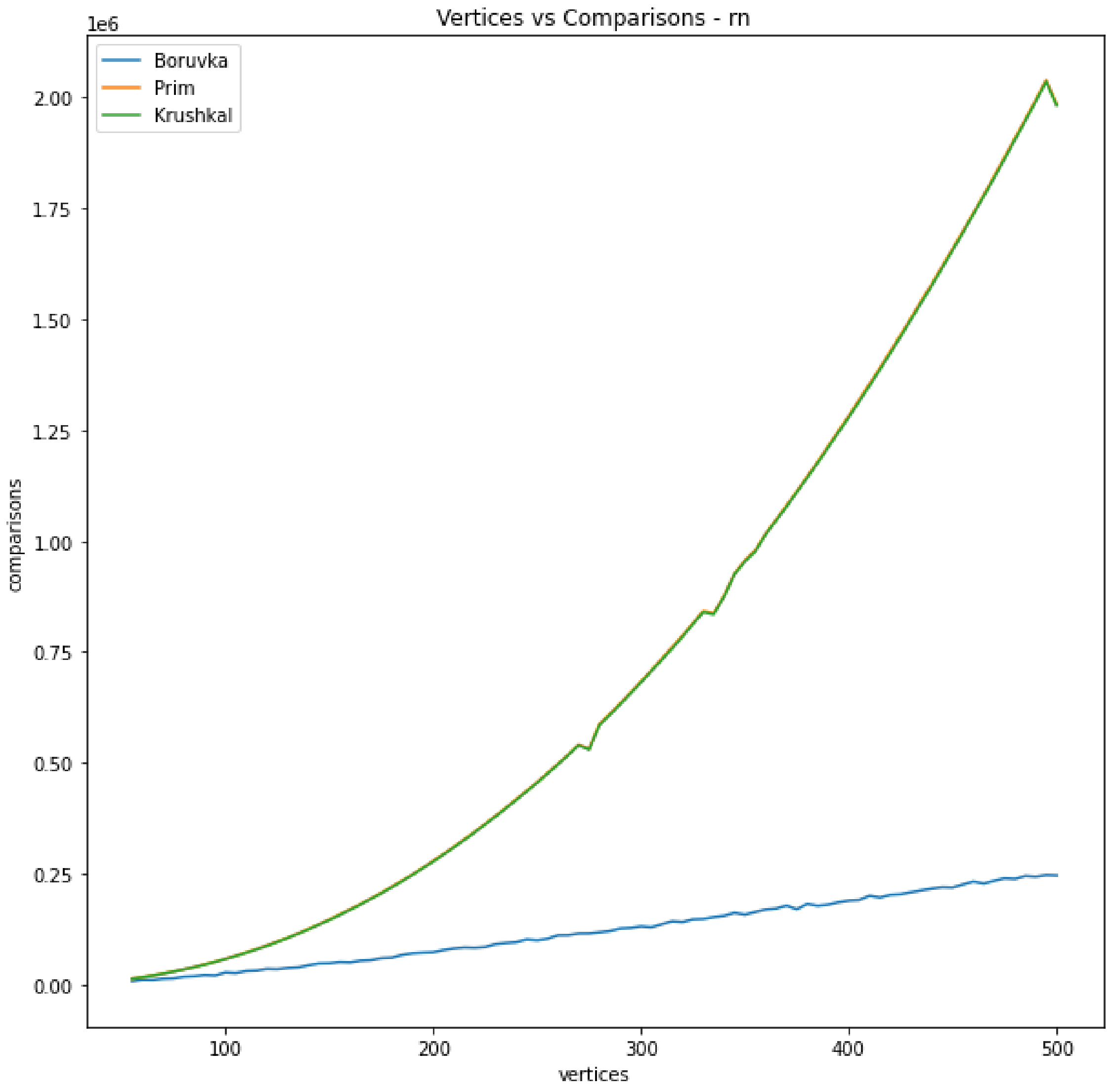


Figure 12: All algorithms - rn graph - vertices vs comparisons

We can see that Boruvka's performs much better compared to other two algorithms when we use rn graphs. Prim's and Krushkal's pretty much have the same number of comparisons across the graph.

3. **Edges vs Comparisons:**

This experiment tested the input graphs generated by varying the number of edges from the minimum possible value to the maximum possible value for the number of vertices as 450. Such input graphs were generated for each type of graph, i.e. rn, gm, gw.

These graphs were then given as inputs to all the three algorithms, Prims, Krushkals, and Boruvkas, and various analyses were conducted based on the output.

(a) Boruvka's Algorithm: For all type of graphs, varying number of edges from minimum to maximum for V=450

We can see that Boruvka's algorithm is performing in a parabolic manner with the number of comparisons by the graphs generated in random mode, and is performing close to linear with graphs generated by geometric mode and geometric wrap-around mode.

Therefore, we can say that Boruvka's algorithm prefers inputs created by a Geometric and geometric wrap-around data generator.

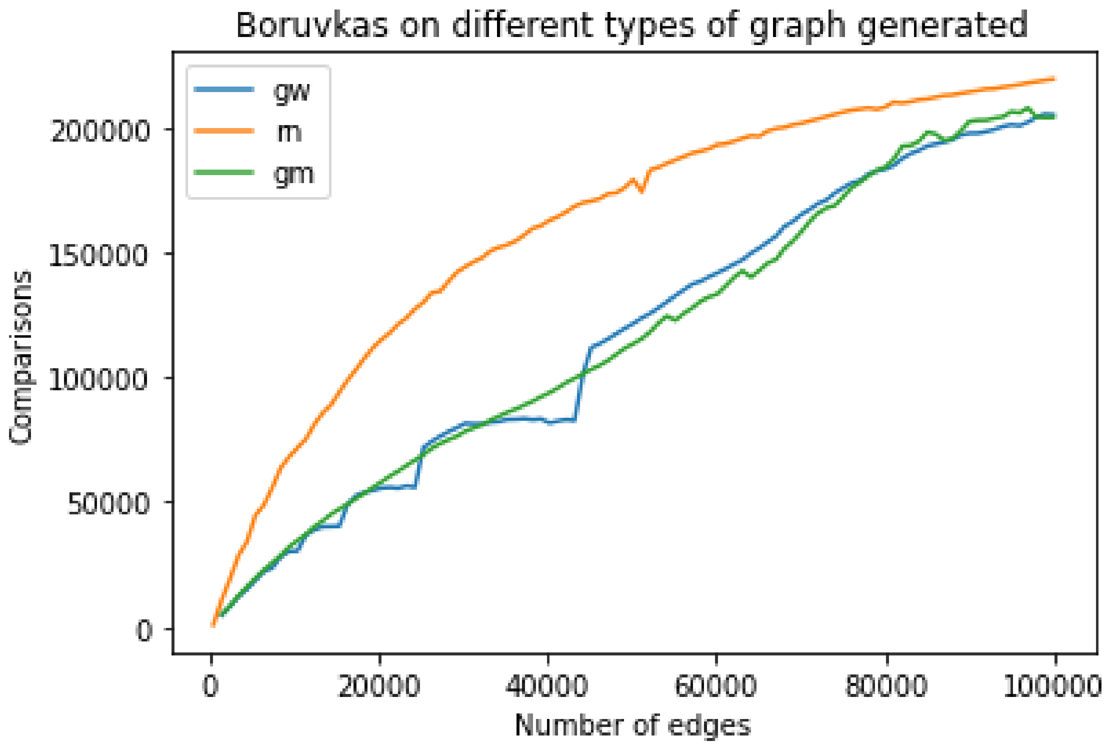


Figure 13: Boruvka's: Varying number of edges from minimum to maximum for V=450

(b) Krushkal's Algorithm: For all type of graphs, varying number of edges from minimum to maximum for V=450

We can see that Krushkal's algorithm is performing in a parabolic manner with the number of comparisons by the graphs generated in geometric mode, and is performing close to linear with graphs generated by geometric mode and geometric wrap-around mode.

Therefore, we can say that Krushkal's algorithm prefers inputs created by a Geometric data generator.

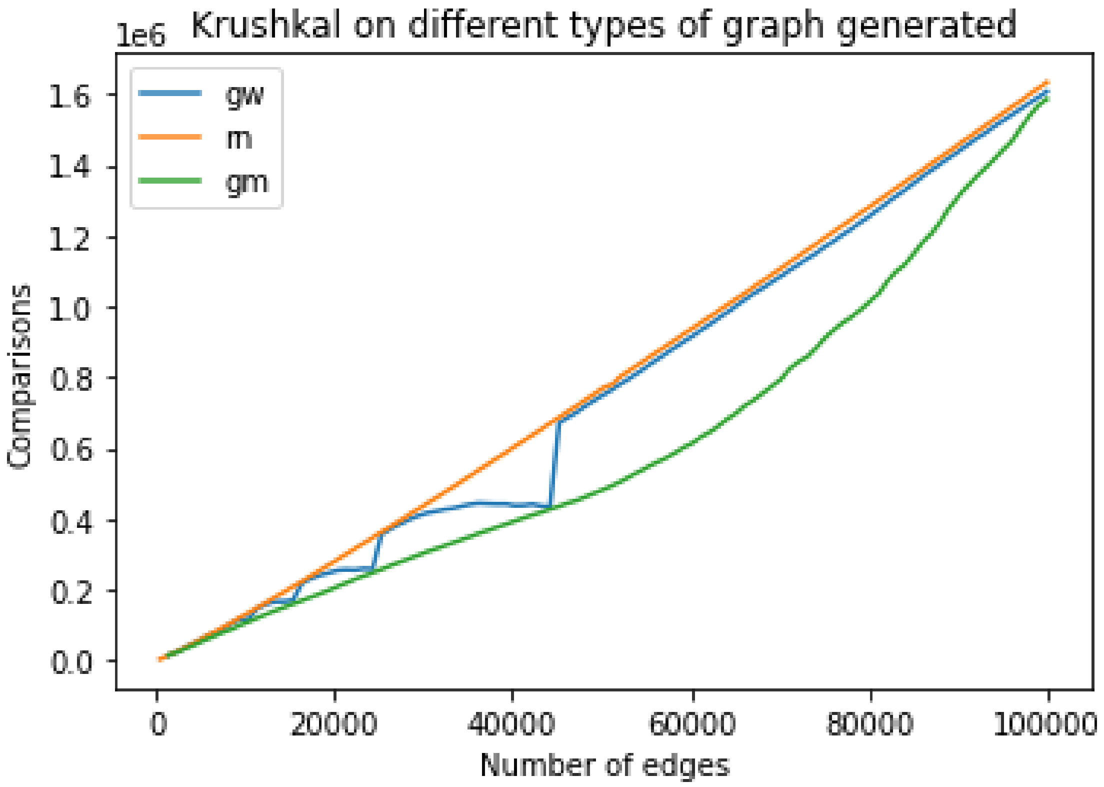


Figure 14: Krushkal's: Varying number of edges from minimum to maximum for V=450

(c) Prim's Algorithm: For all type of graphs, varying number of edges from minimum to maximum for V=450

We can see that Prim's algorithm is performing in a parabolic manner with the number of comparisons by the graphs generated in geometric mode, and is performing close to linear with graphs generated by geometric mode and geometric wrap-around mode.

Therefore, we can say that Prim's algorithm prefers inputs created by a Geometric data generator.

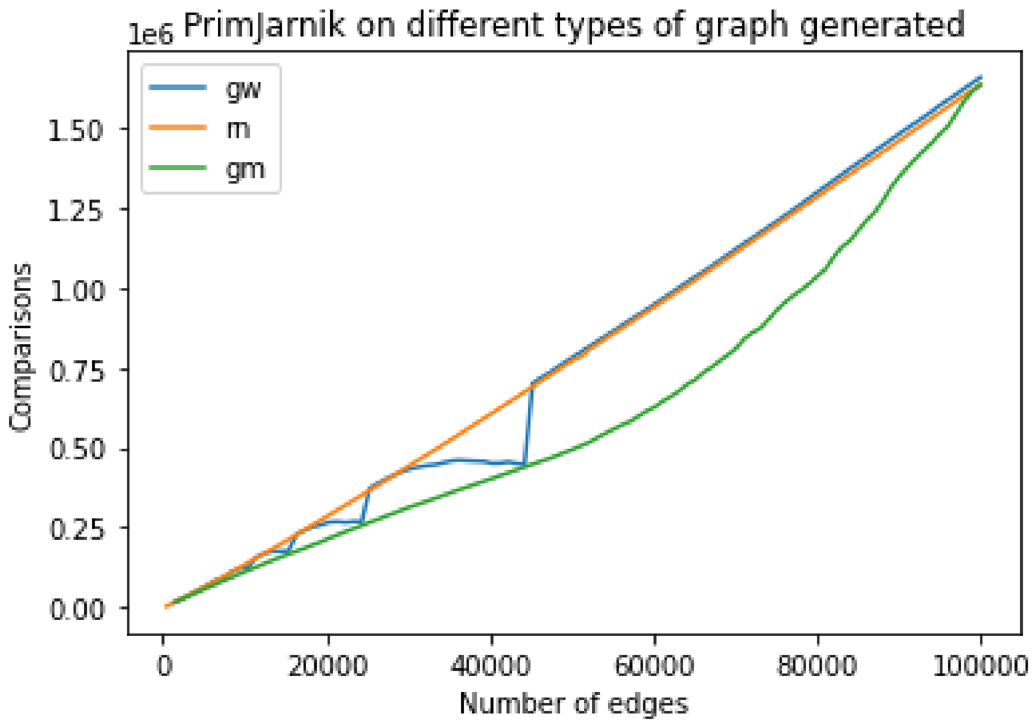


Figure 15: Prim's: Varying number of edges from minimum to maximum for V=450

(d) All Algorithms - gm graph, showing edges vs comparisons

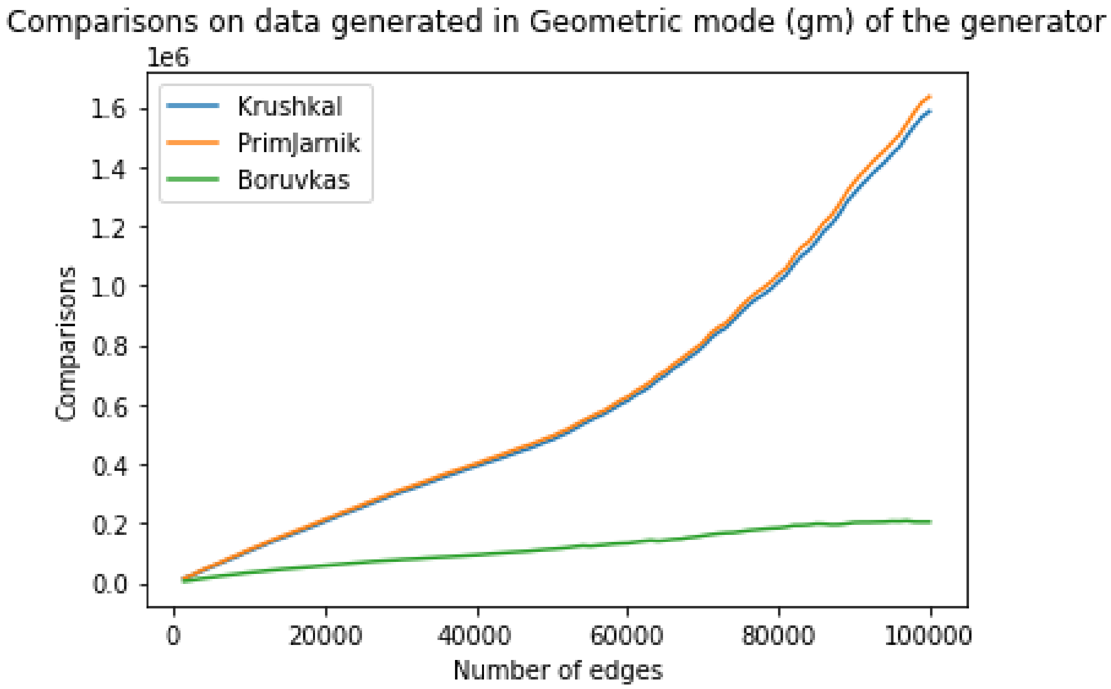


Figure 16: All algorithms - gm graph - edges vs comparisons

(e) All Algorithms - gw graph, showing edges vs comparisons

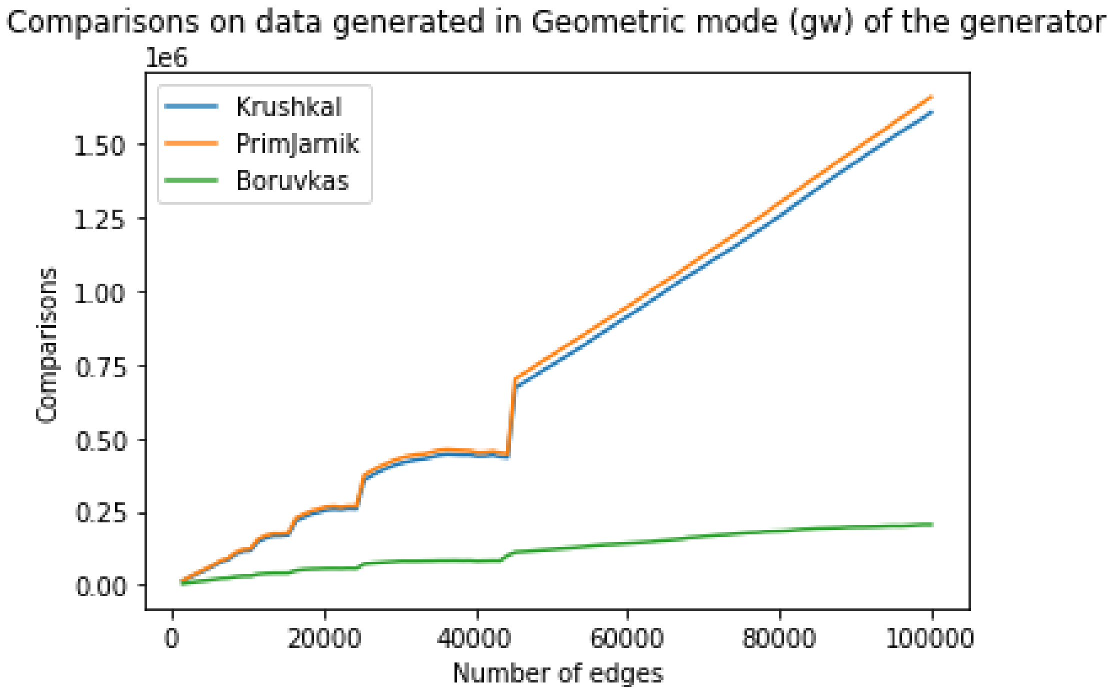


Figure 17: All algorithms - gw graph - edges vs comparisons

(f) All Algorithms - rn graph, showing edges vs comparisons

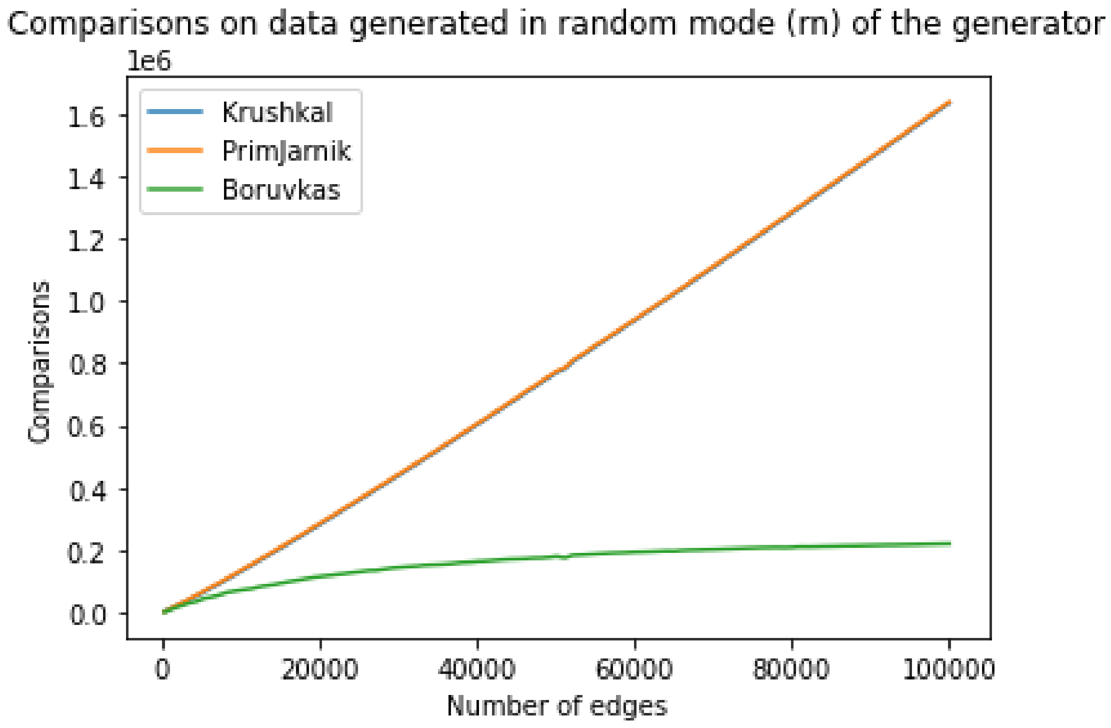


Figure 18: All algorithms - rn graph - edges vs comparisons

6 Conclusion

After performing various comparisons based on a number of scenarios for the three algorithms: Boruvka, Krushkal and Prim Jarnik we draw the conclusion that while Boruvka performs better in terms of comparisons, Krushkal takes the lead when considering runtime. We can also conclude that . All three algorithms have been implemented in a stable fashion and this is proved by the fact that with constant input, running the algorithm yields the same number of comparisons.