

# CARPOOLING ALGORITHM FOR DECREASE THE ENVIRONMENTAL IMPACT.

Felipe Sosa Patiño  
Universidad Eafit  
Colombia  
[fsosap@eafit.edu.co](mailto:fsosap@eafit.edu.co)

Camila Barona Cabrera  
Universidad Eafit  
Colombia  
[cbaronac@eafit.edu.co](mailto:cbaronac@eafit.edu.co)

Mauricio Toro  
Universidad Eafit  
Colombia  
[mtorobe@eafit.edu.co](mailto:mtorobe@eafit.edu.co)

## ABSTRACT

The problem is the traffic at the streets, because some people use their cars alone, then the number of cars is very big, and the city get overload of cars, their noise and their contamination. This situation has to be solved because the Medellin city has constant alarms for big levels of contamination and pollution at the air. Besides the respiratory diseases that the contamination can generate, other aspects of the quality of life are affected: the people get late to their jobs, to their schools or college, and get late to other important dates. This is because the public transport is usually very crowded and uncomfortable.

## KEY WORDS

Theory of computation → Design and analysis of algorithms → Sorting and Searching → DataSets

## 1. INTRODUCTION

The arrival of the personal cars and motorcycles made easier the way the people moves from one place to another, but the overcrowding of that vehicles at actual times, has been an obstacle for the mobility through the city, because in most cases every person has his own car and uses it alone.

## 2. PROBLEM

Each person at his or her car or motorcycle in a big city of almost 2'530.000 people, makes a chaos at the streets, because there are some accidents starred in most of the times by motorcycles because those vehicles are less balanced than the cars. Those accidents produce mobility stagnation and a lot of stress on people.

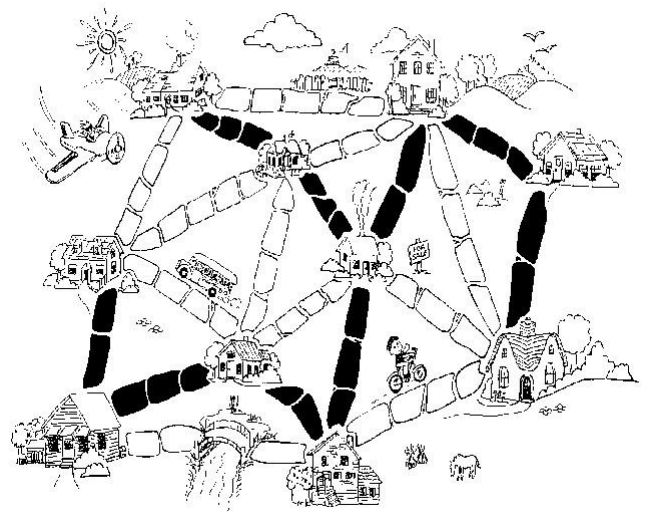
## 3. RELATED WORK

For the current report, an investigation was conducted of different data structures that promoted to give an effective solution to the problem posed.

### 3.1 Muddy City

This problem is based on enough streets must be paved so it is possible to travel from any house to any other house, possibly via other houses. The paving should be done at a minimum cost. For that, a possible solution is **minimum spanning tree (MST)**

The 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.



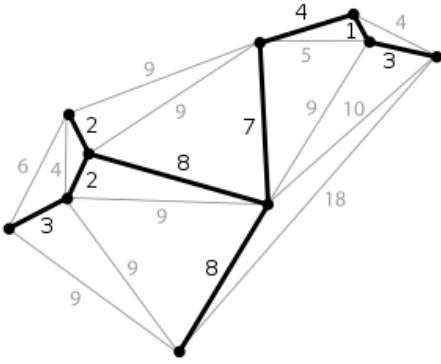


Figure 2.

### 3.2 Deliveries problem

This problem is based on find the best route to make deliveries to all the places marked in the map, starting from a special company saved time and money the shortest route may be found by trial and improvement methods, or by considering all the possible routes.

**Prim's Algorithm** is a possible solution of this problema. Is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. The algorithm operates by building this tree one vertex at a time from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

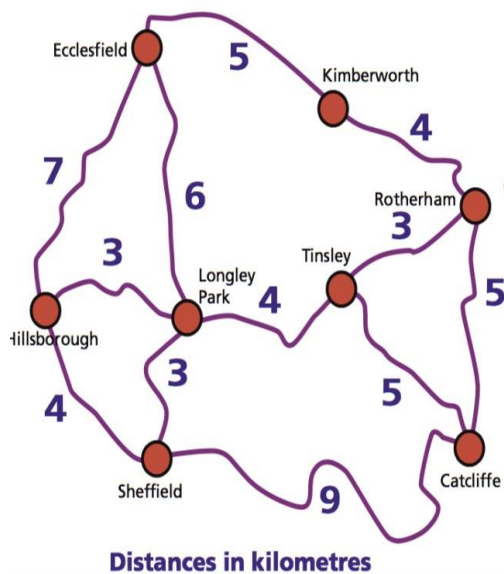


Figure 3.

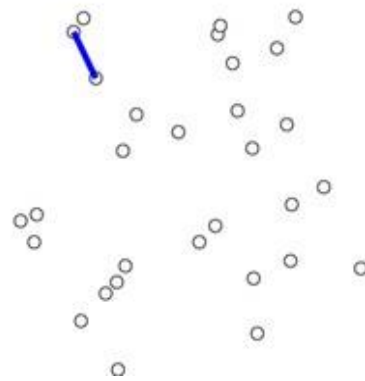


Figure 4.

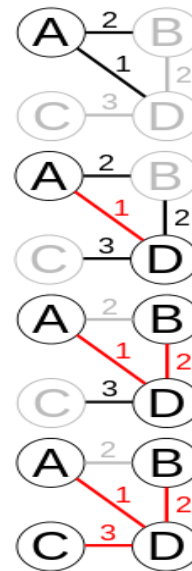


Figure 5.

**3.3 Network design problem** This problem is about how a company with different offices can connect a phone lines with each other with the minimum total cost.

One of the many solutions of this problem is **Kruskal algorithm**: is an algorithm of graph theory to find a minimum covering tree in a connected and weighted graph. It looks for a subset of edges that forming a tree, include all vertices and where the value of the sum of all the edges of the tree is the minimum. If the graph is not connected, then look for a minimum expanded forest (a minimum expanded tree for each related component).

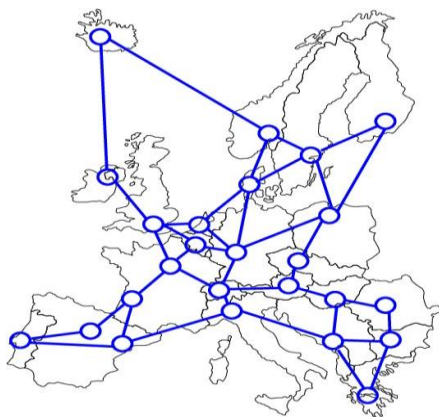


Figure 6.

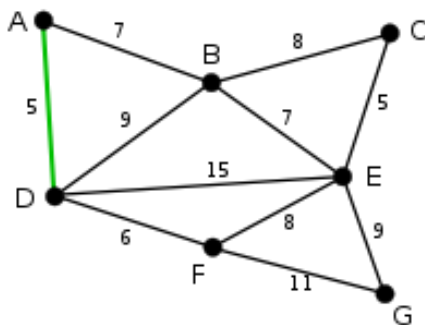


Figure 7.

### 3.4 Traveling salesman problem

The problem of the traveling salesman, problem of the traveling salesman, problem of the traveling agent or traveling salesman problem (TSP) answers the following question: given a list of cities and the distances between each pair of them, which is the shortest route possible that visits each city exactly once and at the end returns to the origin city?

One of the solutions for this problem is: **Boruvka algorithm**. The algorithm begins by examining each vertex and adding the lower weight arc from that vertex to another one in the graph, without taking into account the already added arcs, and continues uniting these groups in the same way until a tree that covers all the vertices.

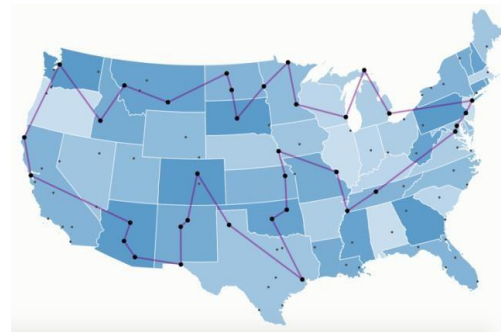


Figure 8.

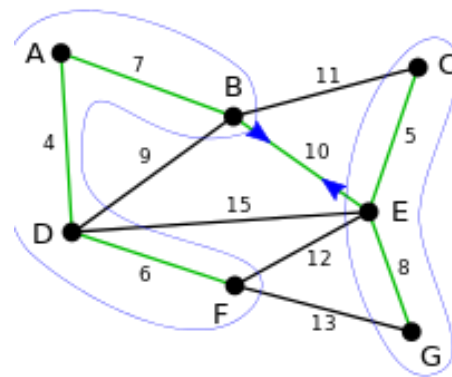


Figure 9.

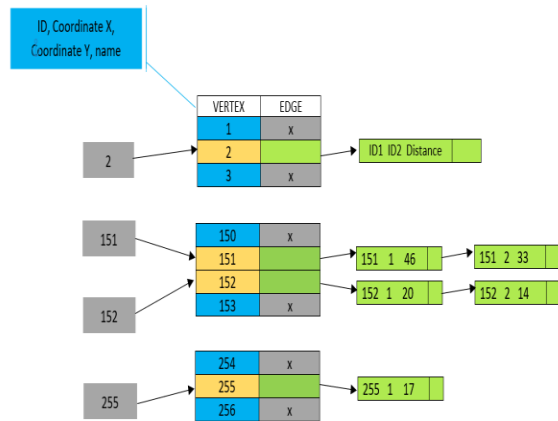
## 4. HASHMAP TO IMPROVE STREET'S MOBILITY

Next, we explain the data structure and the algorithm.

### 4.1 Data structure

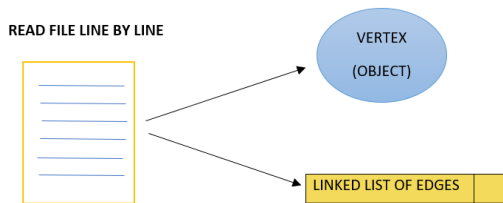
The Data structure we're using is a really big hash map. Why so big? Because that hash map saves all the information given by the Data sets. How do we do that? We use the vertex ID (Identification) which is a parameter, as key of the hash map, and just with the vertex ID we're able to get :

1. The information of the vertex (name, x coordinate, y coordinate)
2. The list of Edges that connect the vertex with others, and the weight of go from one to another.

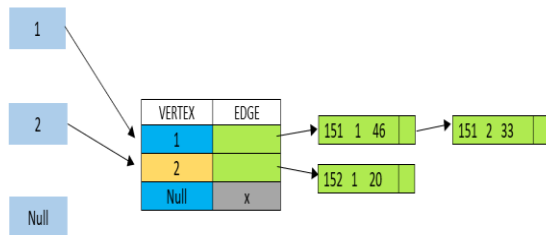


Graphic 1: HashMap that contains a key and a pair with vertex (ID, Coordinate X, Coordinate Y, Name) and edge (ID1, ID2, Distance)

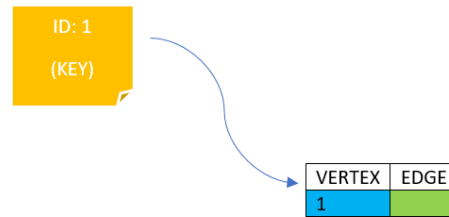
#### 4.2 Data structure operations



Graphic 2: Image of read file line by line (When we get the information we create an object of vertex and a linked list of edges)



Graphic 3: Image of method put in a HashMap.



Graphic 4: Image of method get in a HashMap.

#### 4.3 Design criteria of the data structure

We use that data structure because we think is the fastest way to save the information of the project. Why? Because in our research about the complexity of the hashmap, in average, of his operations search, insertion and delete, is  $O(1)$  and that's amazing. Because we have to work with a big amount of information, about vertices (people) and edges (routes). So the best way to save all that information is with the hash map and in case we need the edges of any vertex, we can also access to that information in  $O(1)$ . And the memory complexity is  $O(n)$  and that's good too because we only reserve space for each vertex, and that's what we need for relating a vertex with other with an edge.

#### 4.4 Complexity Analysis

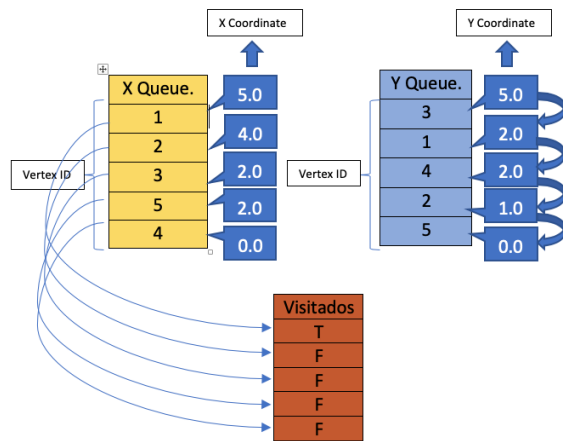
Method	Complexity
MakeMap	$O(n)$
GetSuccessors	$O(v)$
Get Weight	$O(v)$

**Table 1:** Complexity of operations of the data structure

#### 4.5 Algorithm.

The algorithm we're going to implement is going to take the hashmap of vertices and edges, and will put the vertices in 2 priority queues, the first one is for sort the values of the x coordinate from each vertex, and the second if for sort the values of the y coordinate from each vertex. Why? Because if we know the extreme values, i mean, the values that are further than others have the priority, because the person that is located there must be the person that pick up people. So, we select the extreme values as the first node of the path. Also we need a boolean array for ask which vertices have been visited. While the algorithm

travel by the successors of the extreme vertex we have mentioned, he is going to select from the successors list his closest vertex having as reference the position of the vertex on the queues and the position of the successor we are comparing. If the successor is close to the vertex, and the path to the destiny don't gets a big changes in time and distance, we choose that successor for being piked up by the person correspondent to the fist edge, and so on, until we relate 5 vertex to the original, marking each vertex selected as visited. After that, we do the same thing whit the next vertex at the queue (any of them) if that vertex isn't visited. That's because if the vertex is visited, we don't need to compare it for select it again.



**Graphic 4:** the way the algorithm tooks information and organize it.

#### 4.6 Calculation of algorithm complexity

Sub problem	Complexity
Creating the graph	$O(V)$
Get Successors	$O(E)$
Get Weight	$O(E)$
Search	$O(V \log V)$
<b>Total complexity</b>	$O(V \log V)$

**Table 2:** complexity of each of the sub-problems that make up the algorithm. Let  $V$  be the vertex and  $E$  the edges.

#### 4.7 Algorithm design criteria

We desingned that algorithm because the way we're going to search won't repite information, because the process made for create paths for 5 people for each extreme vertex, isn't going to admit another vertex, so those verteces we've visited, won't be processed again. We think that the previus explanation shows how fast the algorithm can be, doing the hole process around  $O(n)$  because it has to visit all the verteces, without repeat, but we have to count with the speed of the processes made in the priority queues. That is going to make our algorithm a little bit slower.

#### 4.8 Execution time

	<i>Data set 1</i>	<i>Data set 2</i>	<i>Data set 3</i>
<i>Mejor caso</i>	938 ms	1100 ms	2014 ms
<i>Caso promedio</i>	1203 ms	1774 ms	2143 ms
<i>Peor caso</i>	2013 ms	2908 ms	3159 ms

**Table 3:** Execution time of the algorithm with differents datasets.

#### 4.9 Memory

	<i>Data Set 1</i>	<i>Data Set 2</i>	<i>Data set 3</i>
<b>Memory consumption</b>	7,75 MB	7,21 MB	7,75 MB

**Table 4:** Memory consumption of the algorithm with different data sets.

#### 4.10 Analysis of the results

Autocomplete structure	HashMap
Heap space	175 MB
Time of creation	75 ms
Time of lookups	5 ms

**Table 5:** Analysis of the results obtained with the implementation of the algorithm

## REFERENCES

1. Minimum spanning tree,2019. Consulted on march 2 of 2019, Wikipedia, The free encyclopedia. Recovered of: [https://en.wikipedia.org/wiki/Minimum\\_s  
panning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)
2. Toy problems for real world. Consulted on march 2 of 2019. Tenderfoot: Unit 2. Recovered of: [https://www.computingatschool.org.uk/da  
ta/tft/02/04Activity Toy Problems Real  
World.pdf](https://www.computingatschool.org.uk/data/tft/02/04Activity_Toy_Problems_Real_World.pdf)
3. Prim's algorithm,2019. Consulted on march 2 of 2019, Wikipedia, The free encyclopedia. Recovered of: [https://en.wikipedia.org/wiki/Prim%27s\\_alg  
orithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm)
4. Kruskal algorithm,2019. Consulted on march 2 of 2019, Wikipedia, The free encyclopedia. Recovered of: [https://es.wikipedia.org/wiki/Algoritmo\\_de  
Kruskal](https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal)
5. Krustal algorithm, 2012. Consulted on march 12 of 2019. Graphs: Software for construction, edition and graph analysis. Recovered of: [http://arodrigu.webs.upv.es/grafos/doku.php  
?id= algoritmo nkruskal](http://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_nkruskal)
6. Applications of MTS. Consulted on march 2 of 2019. GeeksforGeeks: A computer science portal of geeks. Recovered of: [https://www.geeksforgeeks.org/applications-  
ofminimum-spanning-tree/](https://www.geeksforgeeks.org/applications-ofminimum-spanning-tree/)
8. Traveling salesman problem, 2019. Consulted on march 2 of 2019. Wikipedia, the free encyclopedia. Recovered of: [https://es.wikipedia.org/wiki/Problema\\_del  
viaj ante](https://es.wikipedia.org/wiki/Problema_del_viaj_ante)
9. Boruvka Alforithm, 2017. Consulted on march 2 of 2019. Wikipedia, the free encyclopedia. Recovered of: [https://es.wikipedia.org/wiki/Algoritmo\\_de  
\\_Bor uvka](https://es.wikipedia.org/wiki/Algoritmo_de_Bor_uvka)
10. Figure 1: [https://alyssea84.wordpress.com/2014/02/1  
5/the -muddy-city/](https://alyssea84.wordpress.com/2014/02/15/the-muddy-city/)
11. Figure 2: [https://en.wikipedia.org/wiki/Minimum spa  
nning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)
12. Figure 3: [https://www.computingatschool.org.uk/data  
/tft/02/04Activity Toy Problems Real Wo  
rld.pdf](https://www.computingatschool.org.uk/data/tft/02/04Activity_Toy_Problems_Real_World.pdf)
13. Figure 4: [https://en.wikipedia.org/wiki/Prim%  
27s\\_algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm)
14. Figure 5: [https://en.wikipedia.org/wiki/Prim%  
27s\\_algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm)
15. Figure 6: [https://raweb.inria.fr/rapportsactivite/  
RA2010/re\\_alopt/uid17.html](https://raweb.inria.fr/rapportsactivite/RA2010/re_alopt/uid17.html)
16. Figure 7: [https://en.wikipedia.org/wiki/Kruskal  
%27s\\_algo rithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)
17. Figure 8: [http://examples.gurobi.com/traveling  
salesman-problem/](http://examples.gurobi.com/traveling-salesman-problem/)
18. Figure 9: [https://en.wikipedia.org/wiki/Borůvk  
a%27s\\_alg orithm](https://en.wikipedia.org/wiki/Borůvk_a%27s_algorithm)

e  
4  
.