Vehicle routing problem to minimize the total time to visit a group of clients and taking time to recharge the vehicle

Miguel Angel Zapata Jimenez
Universidad Eafit
Colombia
mazapataj@eafit.edu.co

Santiago Ochoa Castaño
Universidad Eafit
Colombia
sochoac1@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

ABSTRACT
Considering new strategies to find the shortest route for a vehicle fleet that needs to deliver an inventory to a specific destination, would bring benefits in vehicle mobilization. However, it is difficult to achieve that objective because there are different variants of this problem. Each one of them has a specific restriction. Such as limited load capacity, restricted hours, insufficient number of cars, and some other restrictions, which cause many algorithms to be made to satisfy those needs. In this case, backtracking can be a good solution to the problem searching the better route and eliminating the least feasible.

Keywords
Heuristic, Computing Methodologies, Modeling and simulation, Software system structure.

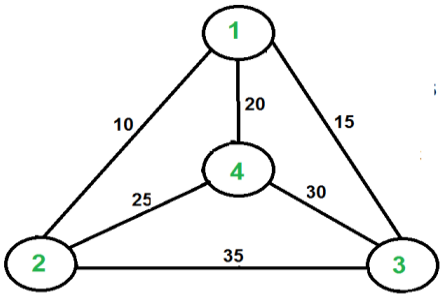
1. INTRODUCTION
Electric vehicles have been a new alternative to reduce the air pollution from petrol or diesel cars. Electricity plays an important role and represents zero emissions making better to the environment. However, the use of electric vehicles for charging and for passenger transport has a limitation: the driving range is limited and the battery charging time is relatively long.

2. PROBLEM
The problem to be solved consists of designing an algorithm to find optimal routes for a group of electric vehicles to deliver merchandise to a set of customers. In this way, the time that takes to visit each customer, recharge the battery, and finish the whole route will be minimized.

3. RELATED WORK

3.1 The Travelling Salesman (TSP)
The travelling salesman consists in finding the shortest route for a person to complete a task given a specific group of destinations. The difficulty of this problem occurs at working with many places to visit because the algorithm must be in charge to verify the least distance or cost for each route.

The famous problem has different solutions. One of them is by using the Brute Force. This method finds the best route by comparing all possible permutations of routes to choose the shortest unique solution. In other words, it calculates the time that it takes to visit each distance, and finally choose the shortest time.



3.2 Profitable Vehicle Routing Problem with Multiple Trips
The Vehicle Routing Problem(VRP) relates two variants which are the Profitable VRP and the VRP with Multiple trips. In relation with the profitability, it consists in the limitations to serve a group of customers due to the budget shortage or for insufficiency of the offer. On the other hand, with relation to the multiple trips, it means that a limited vehicle fleet must perform several routes with a strict schedule.

The problem was solved through using two algorithms based on three-arrays: Hill Climbing algorithm and Variable Neighborhood Descent algorithm.

The Hill Climbing algorithm begins by iterating with an arbitrary solution, then tries a different solution by changing the search parameters. If the change produces a better solution, try a new one repeating the process until no improvements can be found.

The Variable Neighborhood descent algorithm performs a search by exploring distant locations or neighborhoods from the current solution and iterates until there are no further possible improvements.

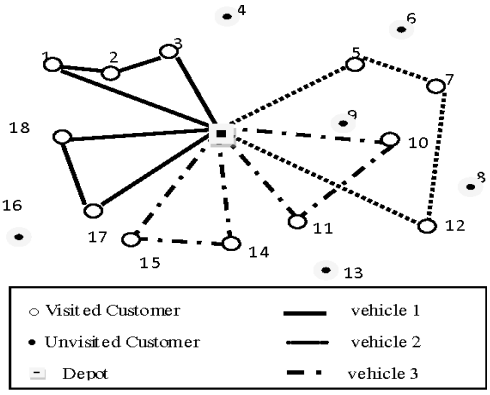
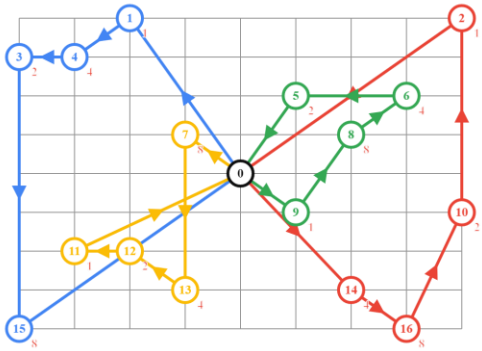


Figure 1. Illustrative Example

3.3 Capacitated Vehicle Routing Problem (CVRP)

It consists in finding a route that does not require excessive consumption of time and resources. The difficulty is given because there is a limited load capacity, and it must return to the main warehouse.

This problem has different solutions, but one of the most famous is the metaheuristics techniques, which is based on using the parameters provided by the user to find an efficient result. In this case, a depth search must be carried out, finding the routes that do not require too much time, in other words, backtracking must be implemented.

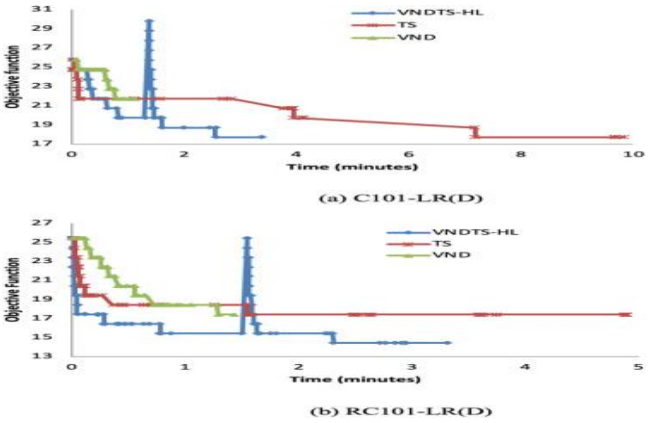


3.4 Heterogeneous vehicle routing problem with time windows and a limited number of resources (HVRPTW-LR):

The problem arises when the available resources like vehicles, drivers, or instruments to serve a subset of customers in a route planning are insufficient. For this reason, each route should be chosen by thinking about minimizing the travel costs and maximizing the total number of served customers.

In the first place, the solution is based in a semi-parallel insertion heuristic. In this way, it is improved by applying the Tabu Search algorithm for the exploration of each route which follows a series of patterns.

The Tabu Search algorithm increases the performance by using memory structures: At the moment of finding a potential solution it is marked as “Tabu”, so the algorithm does not visit that possible solution again.



(a) C101-LR(D)

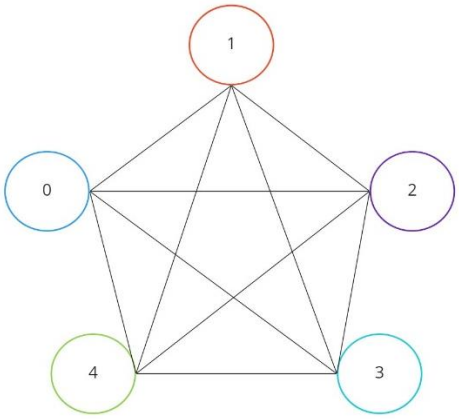
(b) RC101-LR(D)

4. TITLE OF THE FIRST ALGORITHM DESIGNED

In what follows we explain the data structure and the algorithm.

4.1 Data Structure

An adjacence matrix is going to be implemented to represent the map of the city, where each node is placed. The value that are in the rows represent the distance between the nodes. (Figure 1).



Matrix	0	1	2	3	4
0	0	23,42	12,42	42,43	1,23
1	46,5	0	16,8	56	92
2	12,2	11,23	0	76,4	35,3
3	98,23	76,42	23,4	0	23,21
4	64,3	90,42	43,4	94,2	0

Figure 1: Adjacence matrix of places. Where can be seen the distance between two specific nodes.

4.2 Operations of the data structure

Design the oepration of the data structure to solve the problem efficiently. Include one figure to explain each operation.

- **addArc***:* Method of adding a new arc, where each node is represented by an integer and a weight represented by a double is assigned as the distance between two nodes. (Figure 2).

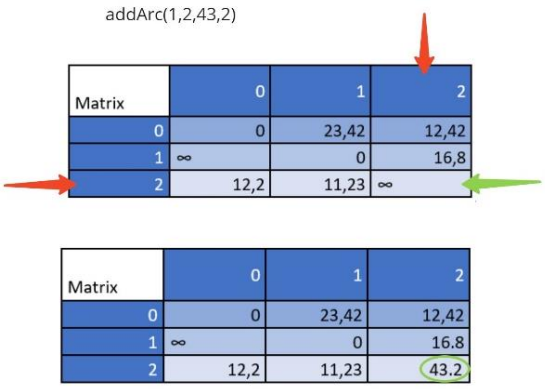


Figure 2: Add a weight between two nodes.

- **getWeight:** Method to obtain the weight or distance between two nodes. (Figure 3).

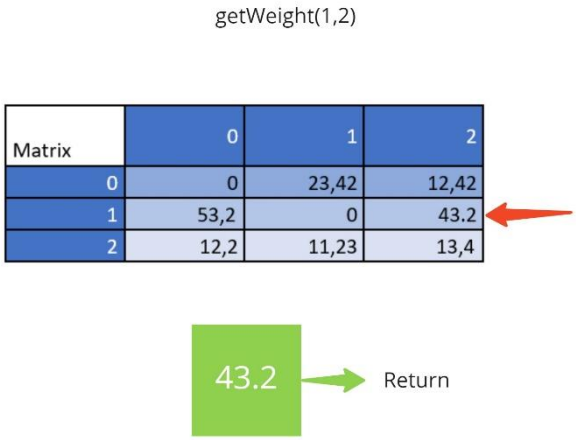


Figure 3: Obtain the weight of two nodes.

- **getSuccessors:** Method to obtain a list of children from a node, that is, all nodes associated to the node assigned as an argument. (Figure 4).

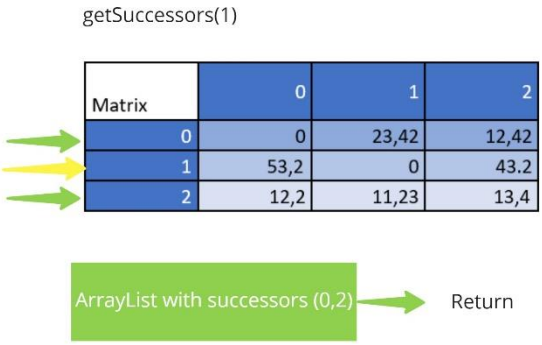


Figure 4: Obtain the successors of a node.

4.3 Design criteria of the data structure

The data structure that was considered to solve the problem was an adjacency matrix. In the first place, it was useful to represent an undirected finite graph which exemplify the city map. The elements of the matrix symbolize whether a pair of vertices are separated by a distance or not, that is, the edges in a graph. The distance value was calculated by finding the distance between two points. Furthermore, it was analyzed the efficiency in time and memory. Although inserting a new vertex or deleting has a complexity of $O(v^2)$, in this case the size of the matrix was specified when initializing, so it is no necessary to perform these operations. On the other hand, it was more important to search a distance with a complexity of O(1). In conclusion, the memory was a priority over time.

4.4 Complexity analysis

Method	Complexity
addArc	O(1)
getWeight	O(1)
getSuccessors	O(n)

Note: The letter n represents the number of nodes.

Table 1: Table to report complexity analysis.

4.7 Algorithm

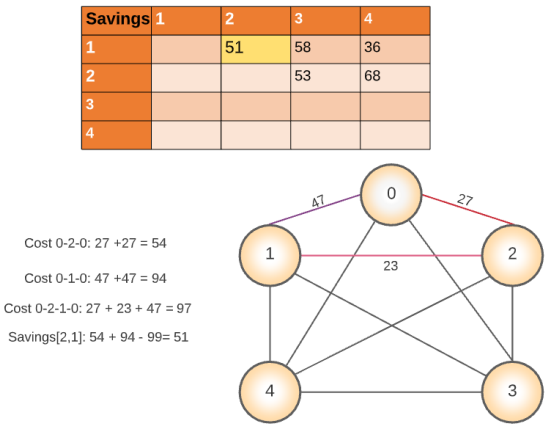


Figure 5: Calculations of savings for each pair of routes.

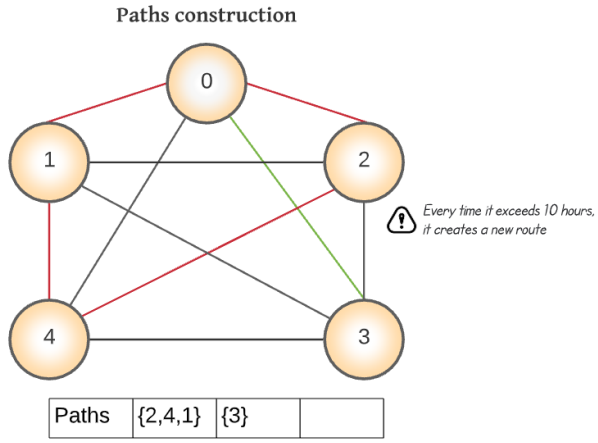


Figure 6: Array list representation to save a new path before exceeding the limit time.

4.6 Complexity analysis of the algorithm

Sub Problem	Complexity
Find the savings and create the matrix	$O(n^2)$
Determinate if a node can be visited	O(n)
Include a node in the path	O(1)
Calculate the path for each vehicle	$O(n^3)$
Total Complexity	**$O(n^3)$**

Table 2: Complexity of each subproblem that is part of the algorithm. N represents the number of nodes in the graph.

4.7 Design criteria of the algorithm

The algorithm that was implemented was based in the Clarke-Wright algorithm. It is known the saving algorithm because it works by calculating the save between two routes to build different paths. The main idea is to determinate the possibility to merge two paths in a single path, after calculating how much time can be saved. The algorithm is beneficial to solve problems where the number of vehicles is not fixed. In this case, it cares more about finding an efficient route under time and battery constraints. Despite it is a heuristic algorithm which does not ensure the best solution, it is efficient to solve the problem without exceeding the maximum response time. Furthermore, during the different operations were used matrixes which allowed to search in O (1).

4.8 Execution times

Compute, execution time for each dataset in the ZIP file

Measure execution time 100 times for each dataset and report average execution time.

	Dataset 1	***Dataset 2***	***Dataset 3***	***Dataset 4***	***Dataset 5***	***Dataset 6***
Best case	4 s	3 s	4 s	5 s	6 s	6 s
Average case	14s	9 s	11 s	12 s	16 s	14 s
Worst case	24 s	16 s	18 s	20 s	26 s	23 s
	Dataset 7	***Dataset 8***	***Dataset 9***	***Dataset 10***	***Dataset 11***	***Dataset 12***
Best case	4 s	3 s	4 s	5 s	6 s	6 s
Average case	13 s	10 s	11 s	14 s	16 s	14 s
Worst case	22 s	18 s	19 s	24 s	26 s	23 s

Table 2: Execution time of the algorithm for different datasets.

4.9 Memory consumption

Measure memory consumption of the algorithm for different datasets

	Dataset 1	***Dataset 2***	***Dataset 3***	***Dataset 4***	***Dataset 5***
Memory consumption	733 KB	825 KB	825 KB	826 KB	825 KB
	Dataset 6	***Dataset 7***	***Dataset 8***	***Dataset 9***	***Dataset 10***
Memory consumption	825 KB	825 KB	825 KB	826 KB	827 KB

		Dataset 11	*Dataset 12*
Memory consumption		824 KB	826 KB

Table 3: Memory consumption of the algorithm for different datasets.

4.10 Analysis of the results

Explain the results obtained. Make a table or a graph explaining the time and memory consumption.

	Memory(KB)	**Execution time**	**Number of clients**	**Number of vehicles**	**Total time**
(1)	755	14 s	320	33	159.52
(2)	826	9 s	320	27	95.97
(3)	825	11 s	320	29	119.29
(4)	826	12 s	320	33	159.52
(5)	825	16 s	320	27	95.97
(6)	825	14 s	320	29	119.29
(7)	825	13 s	320	33	159.52
(8)	825	10 s	320	27	95.97
(9)	825	11 s	320	29	119.29
(10)	826	14 s	320	33	159.52
(11)	825	16 s	320	27	95.97
(12)	826	14 s	320	29	119.29

Table 5: Analysis of the results obtained from the algorithm execution.

5. Vehicle routing problem to minimize the total time to visit a group of clients and taking time to recharge the vehicle.

5.1 Data structures

1.

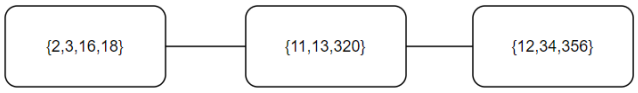


Figure 7. Array List responsible of contain the routes created

2.

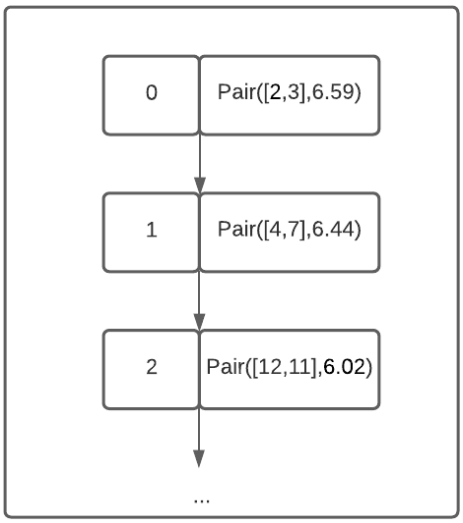


Figure 8: An array list that in each position contains a pair where the key is a pair of nodes, and the value is the distance between those two nodes

3.

Matrix[][]	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	56,21	65,78	73
2	0	0	0	0	34,22	45,33	67,88
3	0	0	0	0	76,32	65,77	53,64
4	0	0	0	0	0	54,78	77,89
5	0	0	0	0	0	0	88,67
6	0	0	0	0	0	0	0

Figure 9. Matrix that contains the distance of a client to a charge station or from a charge station to a other charge station.

5.2 Operations of the data structure

1- savings.get(i).first[0]

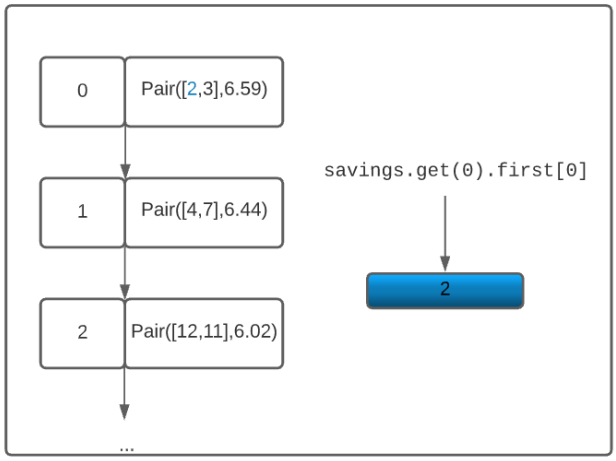


Figure 10: The process to get one of the nodes. The origin or the destiny.

2-savings.get(i).second

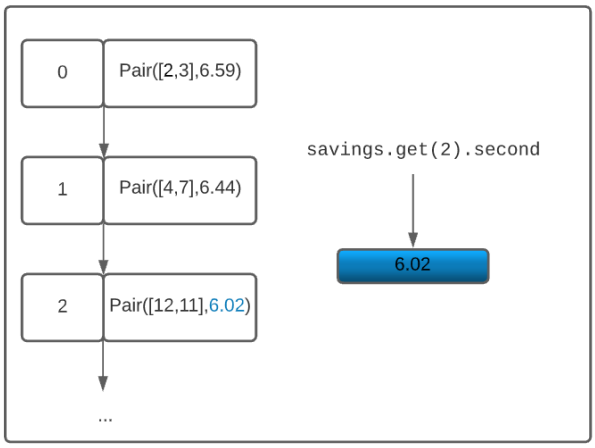


Figure 11: The process to get the distance between the two nodes

3-stations[node][i]

Matrix[][]	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	56,21	65,78	73
2	0	0	0	0	34,22	45,33	***67,88***
3	0	0	0	0	76,32	65,77	53,64
4	0	0	0	0	0	54,78	77,89
5	0	0	0	0	0	0	88,67
6	0	0	0	0	0	0	0

```
stations[2][6] = 67,88
```

Figure 12: The process to get the distance between a client and a charge station.

5.3 Design criteria of the data structure

There are three types of data structures that were implemented to find the best solution, the first one was and Array List that is responsible for saving all the routes once created. Thanks to this a specific route is represented by clients and charge stations being access in O(n). The second one was another Array List which is containing the savings between each node. It allowed to conform pairs where the key is an array of integers size (couple of nodes which are connected), and the value represents the achieved savings. This particularly data structure saves time and memory by avoiding processes to perform the same action. Finally, a matrix implements the charging stations for the solution. It contains the distances between stations with other stations or clients. The matrix is filled in the way of not having repeated values. Thanks to this, at the hour of looking the closest charge station for a node, if it is a client we iterated over row. On the other hand, if it is charge station we iterated over columns.

5.4 Complexity analysis

Array List for routes

Sub problem	Complexity
Get a route	O(n)
Total Complexity	**O(n)**

Table 5: Complexity

Matrix for stations

Sub problem	Complexity
Get a distance betwen node and charge station	O(n*n)
Total Complexity	**$O(n^2)$**

Table 6: Complexity

Array List for savings

Sub problem	Complexity
Get the saving between two nodes	O(n)
Get a node from the pair	O(n)
Total Complexity	**O(n)**

Table 7: Complexity

5.6 Complexity analysis

Sub problem	Complexity
Find the saving and create the matrix	$O(n^2)$
Determinate if a node can be visited	$O(n)$
Include a node in the path	$O(1)$
Calculate the path for each vehicle	$O(n^3)$
Implement charge stations	$O(n^2)$
Total Complexity	**$O(n^3)$**

Table 8: It is showing the complexity to each method used

5.7 Design criteria of the algorithm

The algorithm that was implemented was based in the Clarke-Wright algorithm. It is known the saving algorithm because it works by calculating the save between two routes to build different paths. The main idea is to determinate the possibility to merge two paths in a single path, after calculating how much time can be saved. The algorithm is beneficial to solve problems where the number of vehicles is not fixed. In this case, it cares more about finding an efficient route under time and battery constraints. Despite it is a heuristic algorithm which does not ensure the best solution, it is efficient to solve the problem without exceeding the maximum response time. Furthermore, during the different operations were used matrixes which allowed to search in O (1).

5.8 Execution time

	Dataset 1	**Dataset 2**	**Dataset 3**	**Dataset 4**	**Dataset 5**
Best	177	158	175	243	175
Avarage	604	303	254	314	237
Worst	1032	448	900	798	498
	Dataset 6	**Dataset 7**	**Dataset 8**	**Dataset 9**	**Dataset 10**
Best	189	185	198	234	254
Avarage	288	303	282	286	338
Worst	983	879	1012	794	896

	Dataset 11	**Dataset 12**
Best	195	166
Avarage	254	292
Worst	765	876

Table 9: Execution time of the algorithm for different datasets.

5.9 Memory used

Memory consuption	**Dataset 1 MB**	**Dataset 2 MB**	**Dataset 3 MB**	**Dataset 4 MB**	**Dataset 5 MB**	**Dataset 6 MB**
	1010	954	959	959	954	959
	Dataset 7 MB	**Dataset 8 MB**	**Dataset 9 MB**	**Dataset 10 MB**	**Dataset 11 MB**	**Dataset 12 MB**
	1037	1031	1037	1037	1031	1037

Table 10: Memory consumption of the algorithm for different datasets.

5.10 Result análisis

	Memory(KB)	**Execution time**	**Number of clients**	**Number of vehicles**	**Total time**
(1)	1010	1032 ms	320	53	400.33
(2)	954	448 ms	320	54	402.42
(3)	959	900 ms	320	49	387.31
(4)	959	798 ms	320	50	388.42
(5)	954	498 ms	320	47	377.33
(6)	959	983 ms	320	53	401.33
(7)	1037	879 ms	320	52	399.65
(8)	1037	1012 ms	320	49	397.31
(9)	1031	794 ms	320	49	397.31
(10)	1037	896 ms	320	52	399.65
(11)	1031	765 ms	320	51	398.23
(12)	1037	876 ms	320	54	402.42

Table 11: Memory consumption and tie execution analysis

6. Conclusions

In relation with efficiency the algorithm proved to be effective. The execution time was moderately efficient, but it can be improved by using another way of stablishing to what charge station the truck needs to go after visited a client. On the other hand, memory consumption of the solution was higher because three types of data structures were implemented. So, the consumption was high. The algorithm is a good way of find an optimize solution. However, some improvements should be added to avoid any kind of mistakes in each process that it did.

6.1 Future works

The first thing that we will improve is implemented aleatory. In this way in each execution the algorithm will find a better solution to the previous one. The other thig that we will improve is optimize the method that include the charge stations in a route if it is necessary. The idea that we used is Clark and wright which is useful if the only restriction is the time, but if we implemented the battery restriction it needs to be develop another idea.

REFERENCES

[1] AdobeAcrobat Reader 7, Be sure that the references section text is Ragged Right, Not Justified. http://ww.adobe.com/products/acrobat/.

2. Fischer, G. and Nakakoji, K. Amplifying designers' creativity with domainoriented design environments. in Dartnall, T. ed. Artificial Intelligence and Creativity: An Interdisciplinary Approach, Kluwer Academic Publishers, Dordrecht, 1994, 343-364.

3. Marco, D., & G, L. M. (1997, julio). Ant colonies for the travelling salesman problem (N.o 2). Elsevier. https://doi.org/10.1016/S0303-2647(97)01708-5

4. Chbichib1, A., Mellouli, R., & Habib, C. (2012, julio). Profitable Vehicle Routing Problem with Multiple Trips: Modeling and Variable Neighborhood Descent Algorithm (N.o 2). American Journal of Operational Research. https://doi.org/10.5923/j.ajor.20120206.04

5. Chbichib1, A., Mellouli, R., & Habib, C. (2012, julio). Profitable Vehicle Routing Problem with Multiple Trips: Modeling and Variable Neighborhood Descent Algorithm (N.o 2). American Journal of Operational Research. https://doi.org/10.5923/j.ajor.20120206.04

6. Molina, J., & Salmeron, J. (2020, septiembre). The heterogeneous vehicle routing problem with time windows and a limited number of resources (N.o 94). Elsevier. https://doi.org/10.1016/j.engappai.2020.103745