ALGORITHM USED TO FIND OPTIMAL ROUTES FOR ELECTRIC VEHICLES

Maria Paulina Ocampo Duque
University Eafit
Colombia
mpocampod@eafit.edu.co

Jose Manuel Fonseca Palacio
University Eafit
Colombia
jmfonsecap@eafit.edu.co

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

ABSTRACT

The objective of this project is to design an algorithm that finds optimal routes to electric vehicles, in this specific case to make more efficient the way of distribute merchandise. This in order to solve the problem which is presented in the batteries of the cars for having a limited durability range and a very long loading time.
Our solution is to create an algorithm where it can analyze piece by piece each route and returns the efficient one.

Keywords

Shortest path, electric cars, efficiency, distance, low cost, Graphs, time

ACM CLASSIFICATION Keywords

Ccs → Theory of computation → Design and analysis of algorithms → Graph algorithms analysis → Shortest paths

1. INTRODUCTION

In the following report it is intended to solve the problematic that companies have when delivering products with electrical cargo trucks, for them to be able to deliver them in the most efficient way, taking into account working hours, number of cars and the lifespan of the car battery. Solving this problem would be able to improve deliveries and assure that companies will use electric cars, even with the need of charging them, allowing a reduction in the carbon emissions released by this cargo trucks.

2. PROBLEM

The problem is based on the low efficiency that electric car batteries have to distribute merchandise, because they take a long time to make the routes due to the battery is constantly unloaded. This has a high impact on society because if it is not solved, transporters would be forced to use other vehicles more harmful to the planet, that is why to prevent that from happening it is necessary to solve this problem.

3. RELATED WORK

3.1 Dijkstra´s algorithm

Shortest path problem is a way to find the shortest possible distances between two vertices on a graph such that the sum of the weights of its constituent edges is minimized, for that reason Dijkstra´s algorithm can help us because is an algorithm that we can use to find the shortest distances or minimum costs depending on what is represented in a graph. Through following steps:
-Start at the ending vertex by marking it with a distance of 0
-Identify all of the vertices that are connected to the current vertex with an edge
-Label the current vertex as visited by putting an X over it
-Of the vertices you must marked, find the one with the smallest mark, and make it your current vertex.
-Once you've labeled the beginning vertex as visited, stop. [1]

3.2 Random walk algorithm

The problem in this algorithm is to find, after some fixed time, the probability distribution function of the distance of the point from the origin. So it solution is the Random walk algorithm, which is a process for determining the probable location of a point subject to random motions, given the probabilities of moving some distance in some direction. [2]

3.3 The traveling salesman problem

The traveling salesman problem consists you finding the shortest path that visits each possible city and returns to the original city. [3]

The problem has multiple solutions and is a NP- hard problem since the number of possibilities for the solution when given a big number of cities is immense. Due to that solving it through brute Force is not recommended but to find an optimization method. Solutions to this type of problem are starting to be solved by swarm type algorithms inspired in bees, where the traveling variable (the salesman in this problem) has certain Independence yet a communication with the hive. [4]

3.4 Floyd-Warshall algorithm

The Floyd-Warshall algorithm is a problem that looks to find the shortest path between every pair of vertices in a given edge weighted directed Graph. The condition for the problem to work is that it must be as mentioned before a weighted graph, it can be directed or undirected, but it does not work with negative cycles (The sum of the edges in the cycle is negative). Such algorithm can be of help since for a graph made of arrays it would find the fastest, or cheapest path from each vertex in the graph. The way it works is that for each vertex it tries all the possible paths including the vertexes it passes through and if its smaller than the specified it is replaced in the matrix.[5][6]

4. FIRST ALGORITHM

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

4.1 Data Structure

We are going to use an undirected graph to represent a map where the client and the station will be shown. And an arraylist to save the routes.

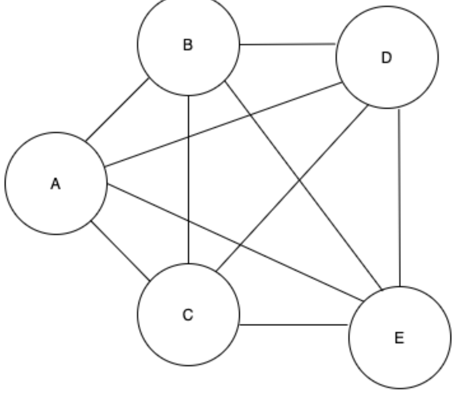


Figure 1: each node contains a client and a station.

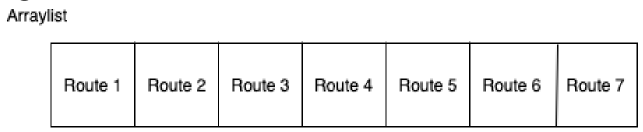
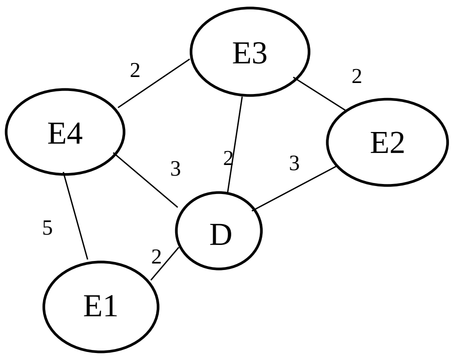


Figure 2: each route contains the distances

4.2 Operations of the data structure

The program finds the closest route from the delivery station and goes there, the same process repeats from node to node, until the amount of time wasted (here represented as the weight of the node) is enough to go back to the station. The operation would be getWeight and then choose the closest.



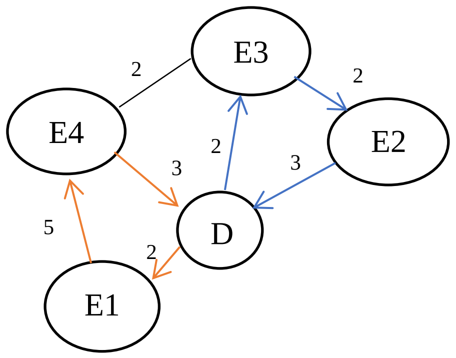


Figure 3: Find the best route and the amount of cars it can take to deliver to all clients and get back to the station. The second drawing represents the two routes taken by two cars and in he order the take them.

4.3 Design criteria of the data structure

The data structure that we use was an adjacent matrix graph to make the map, due to the time has a priority here because has a complexity O(1) it's the most efficient way to do operations while the map is being created.
To save the routes we use an arraylist because with that we do not need to go through all the lists, but it's stored at once each route in one.

4.4 Complexity analysis

Method	Complexity
Matrix graph	O(1)
Arraylist	O(N)
Total complexity	O(N)

Table 1: Table to report complexity analysis.

4.5 Execution time

	Dataset 1	***Dataset 2***	***Dataset 3***	***Dataset 4***
Best case	0.004 s	0.004 s	0.004 s	0.004 s
Average case	0.00705 s	0.00676 s	0.00655 s	0.00636 s
Worst case	0.016 s	0.014 s	0.014 s	0.015 s

Table 2: Execution time of the data structure for different datasets.

4.6 Memory consumption

	Dataset 1	*Dataset 2*	*Dataset 3*	*Dataset 4*
Memory usage	1.597 MB	1.596 MB	1.596 MB	1.596 MB

Table 3: Memory used for each operation of the data structure and for each data set data sets.

4.7 Result analysis

The program was tested with four different datasets, the results are quite similar since all the datasets have the same, number of clients and as for now the program doesn't take into account the battery usage of the car, the slight difference in time and memory is due to the different possible routes the cars can take.

	Name	Num. Clients	Time	Memory usage
Dataset 1	tc2c320s24cf0.txt	320	0.00705 s	1.597 MB
Dataset 2	tc2c320s24cf1.txt	320	0.00676 s	1.596 MB
Dataset 3	tc2c320s24cf4.txt	320	0.00655 s	1.596 MB
Dataset 4	tc2c320s24ct0.txt	320	0.00636 s	1.596 MB

Table 4: Analysis of the results

4.8 Greedy algorithm

We use greedy algorithm because is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy. Besides is not that efficient in terms of memory, in terms of time is really faster.

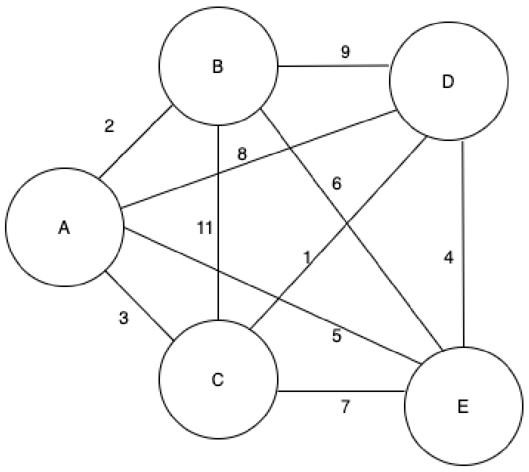


Figure 4: optimal choice of cost in greedy algorithm

4.9 Complexity analysis of the algorithm

Subproblems	**Complexity**
Create the graph	O(n)
Check which nodes are left to visit	O(m)
Find the shortest distance from one node	O(m)
Total complexity	O(n+m^2)

Table 2: Complexity of each subproblem that is part of the algorithm. Let n be the amount of nodes, m the number of clients.

4.10 Design criteria of the algorithm

The algorithm was designed the way it was designed because in terms of time consumption is considerably faster than other options like backtracking or brute force, which might find a better solution, but they are considerably slower, and will possibly occupy more memory, due to the recursive cases in backtracking. This method provides a fast, efficient solution which can also be modified to optimize its answer.

.

5. FINAL SOLUTION DESIGNED

In what follows we explain the data structure and algorithms for this solution.

5.1 Data Structure

For this final solution we use the same as the first one; an undirected graph to represent a map where the client and the station will be shown and an arraylist to save the routes.

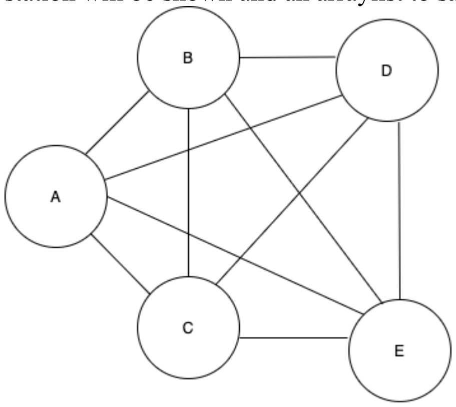


Figure 5: each node contains a client and a station.

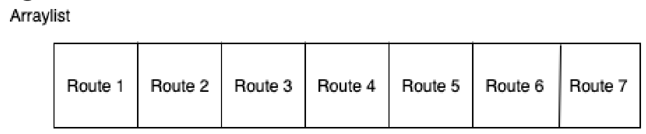
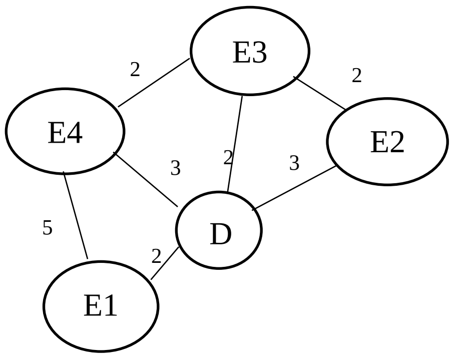


Figure 6: each route contains the distances

5.2 Operations of the data structure

The program finds the closest route from the delivery station and goes there, the same process repeats from node to node, until the amount of time wasted (here represented as the weight of the node) is enough to go back to the station. The operation would be getWeight and then choose the closest.



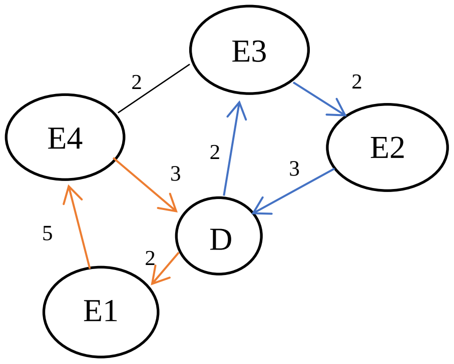


Figure 7: Find the best route and the amount of cars it can take to deliver to all clients and get back to the station. The second drawing represents the two routes taken by two cars in orders the take them.

5.3 Design criteria of the data structure

The data structure that we use was an adjacent matrix graph to make the map, due to the time has a priority here because has a complexity O(1) it's the most efficient way to do operations while the map is being created.
To save the routes we use an arraylist because with that we do not need to go through all the lists, but it's stored at once each route in one.

5.4 Complexity analysis

Method	Complexity
Matrix graph	O(1)
Arraylist	O(N)
Total complexity	O(N)

Table 5: Table to report complexity analysis

5.5 Execution time

	Dataset 1	***Dataset 2***	***Dataset 3***	***Dataset 4***
Best case	0.004 s	0.004 s	0.004 s	0.004 s
Average case	0.00705 s	0.00676 s	0.00655 s	0.00636 s
Worst case	0.016 s	0.014 s	0.014 s	0.015 s

Table 6: Execution time of the operations of the data structure for different datasets.

5.6 Memory used

Report the memory used for each data set

	Dataset 1	***Dataset 2***	***Dataset 3***	***Dataset 4***
Memory usage	1.597 MB	1.596 MB	1.596 MB	1.596 MB

Table 7: Memory used for each operation of the data structure and for each data set data sets.

5.7 Result analysis

The program was tested with four different datasets, the results are quite similar since all the datasets have the same, number of clients and as for now the program doesn't take into account the battery usage of the car, the slight difference in time and memory is due to the different possible routes the cars can take.

	Name	Num. Clients	Time	Memory usage
Dataset 1	tc2c320s24cf0.txt	320	0.00705 s	1.597 MB
Dataset 2	tc2c320s24cf1.txt	320	0.00676 s	1.596 MB
Dataset 3	tc2c320s24cf4.txt	320	0.00655 s	1.596 MB
Dataset 4	tc2c320s24ct0.txt	320	0.00636 s	1.596 MB

Table 8: Analysis of the results

5.8 Algorithm

We use greedy algorithm because is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy. Besides is not that efficient in terms of memory, in terms of time is really faster.

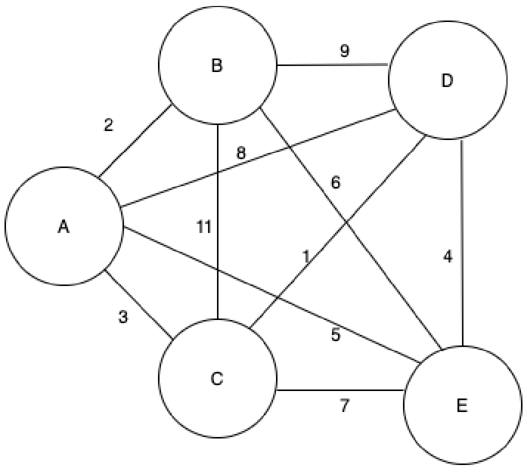


Figure 8: optimal choice of cost in greedy algorithm

5.9 Complexity analysis of the algorithm

Subproblems	**Complexity**
Create the graph	O(n)
Check which nodes are left to visit	O(m)
Find the shortest distance from one node	O(m)

Table 8: Complexity of each subproblem that is part of the algorithm. Let n be the amount of nodes, m the number of clients.

5.10 Design criteria of the algorithm

The algorithm was designed the way it was designed because in terms of time consumption is considerably faster than other options like backtracking or brute force, which might find a better solution, but they are considerably slower, and will possibly occupy more memory, due to the recursive cases in backtracking. This method provides a fast, efficient solution which can also be modified to optimize its answer.

.

5.11 Execution times

	Dataset 1	***Dataset 2***	***...Dataset n***
Best case	0.0016 s	0.0016 s	0,0054s
Average case	0.017 s	0.0309 s	0.0253 s
Worst case	0.0337 s	0.0293 s	0.0452s

Tabla 9. Execution time of the operations of the data algorithm for each data set.

5.12 Memory consumption

Report the memory used for each data set

	Dataset 1	***Dataset 2***	***Dataset 3***
Memory consumption	5 MB	4.9 MB	5 MB

Tabla 10. Memory used for each operation of the algorithm for each data set data sets.

5.13 Analysis of the results

Execution time	Node number	Memory consumption
0.00286 s	1	5 MB
0.0544 s	130	8 MB
0.36 s	482	12 MB

Table 11. Analysis of the results

6. CONCLUSIONS

This project was analyzed in terms of efficiency, for that reason we chose an algorithm faster like Greedy algorithm even though this algorithm is not the most optimal one, meets the requirements of time and memory consumption.
We could implement a recursive data structure to make it more optimal and try another algorithm because we saw that this one is a little slow in terms of memory compared with terms of time.
But for a first version we think that we made a good job and we can improve it in the future in this work, or others because with this project we learned many important knowledges like the importance of time and memory, the differences between algorithms and how apply them.

6.1 Future work

In the future we want show the generated routes in a graph, to make it more dynamic and also make our code more optimal and recursive, because with that, the complexity in terms of time and consumption will be better.

ACKNOWLEDGEMENTS

We want to thank Simón Marín, Engineering student at EAFIT university and Data structures and Algorithm's instructor who explained and support us with the concepts related with the project and the course

REFERENCES

[1] Pennington, L., n.d. *Dijkstra's Algorithm: Definition, Applications & Examples*. [video] Available at: https://study.com/academy/lesson/dijkstra-s-algorithm-definition-applications-examples.html
[Accessed 14 February 2021].
[2] Encyclopedia Britannica. 2008. *Random walk | mathematics and science*. [online] Available at: https://www.britannica.com/science/random-walk
[Accessed 14 February 2021].
[3] *Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming) - GeeksforGeeks*. *GeeksforGeeks.com*, 2018.
Available at: https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/
[Accessed 15 February 2021].
[4] Nunes de Castro, L. and A. S. Masutti, T. *hindawi.com*, 2017. *Bee-Inspired Algorithms Applied to Vehicle Routing Problems: A Survey and a Proposal.* [online] Available at: https://www.hindawi.com/journals/mpe/2017/3046830/
[Accessed 15 February 2021].
[5] *Floyd-Warshall Algorithm*. [online] Available at: https://www.programiz.com/dsa/floyd-warshall-algorithm
[Accessed 15 February 2021].
[6] GeeksForGeeks. 2021.*Floyd Warshall Algorithm | DP-16 - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/.
[Accessed 15 February 2021]