Optimization of fuel consumption in package delivery

Bruno Rodrigues Faria[1], **Lucas de Paula Vieira Santos**[2], **Maria Luísa Tomich Raso**[3]

[1]Institute of Exact Sciences and Informatics - ICEI (PUC - MG)

Abstract. *This article describes in detail the solution implemented for the package delivery optimization problem, aiming at lower fuel consumption, also known as the 'Vehicle Routing Problem' (VRP), a type of problem similar to the Traveling Salesman Problem (TSP). For this, two different types of algorithms were used, one brute force and the other by branch-and-bound, in addition to a variation of branch-and-bound aiming for efficiency.*

Resumo. *Este artigo descreve com detalhes a solução implementada para o problema de otimização de entrega de pacotes visando menor gasto de combustível, também conhecido como 'Vehicle Routing Problem' (VRP), um tipo de problema similar ao Problema do Caixeiro Viajante (TSP). Para isso, foram utilizados dois tipos diferentes de algoritmos, um de força bruta e o outro por branch-and-bound, além de uma variação do branch-and-bound visando velocidade.*

1. Introduction

Optimization problems play a crucial role in several productive sectors, allowing to find efficient solutions to logistical and distribution challenges. A common example is the planning of cargo transport between different geographical points. The objective of this work is to approach the problem of collecting and delivering products in a chain of stores, using three optimization strategies: brute force, branch-and-bound and alternative branch-and-bound.

In the development of this work, the coordinates of the stores, the maximum load capacity of the truck, the fuel yield and the information about the origins and destinations of the products, all contained in a text file, will be considered.

Through a graphic animation, it is intended to illustrate the sequence of truck trips and the variation of its load between branches, providing a clear view of the routes traveled and the fuel consumption associated with each path.

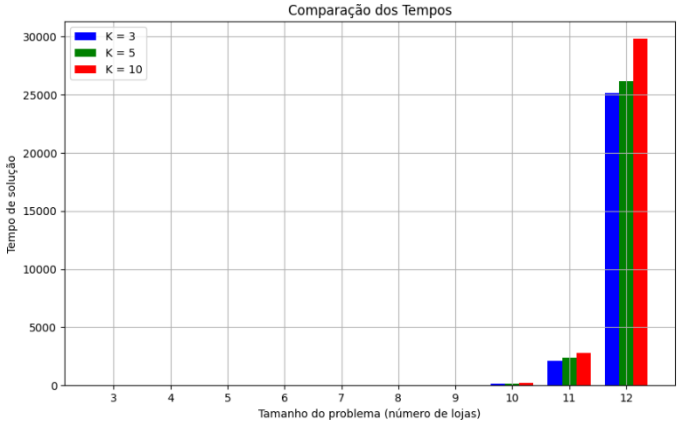
Based on the comparative analysis between brute force, branch-and-bound and alternative branch-and-bound strategies, this study seeks to provide a contribution to the optimization of cargo transport in store chains, highlighting the importance of choosing the method most suitable for the efficient resolution of this type of problem.

2. Proposed solution

2.1. Algorithms used

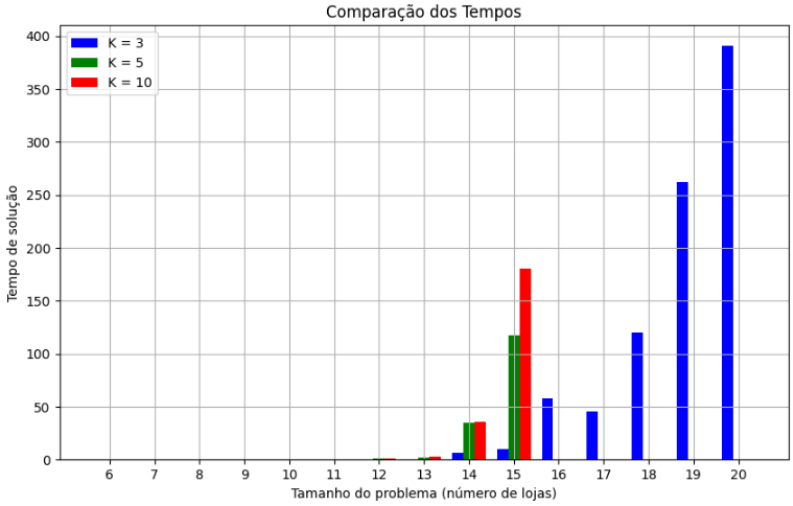
The initial approach consists of implementing a brute force algorithm, which seeks to generate all possible solutions and select the one that results in the lowest fuel consumption, respecting the truck load restrictions. This approach allows to exhaustively examine all path combinations, enabling the identification of the optimal solution, but it can be computationally intensive. With this we conclude that the brute force algorithm has a complexity $\mathcal{O}(n!)$, where **n** is the number of stores different from ***0*** as shown in the graph below, it is observed that it increases very quickly and the other executions are almost impossible to be seen in the plot, due to the complexity:

Figure 1. Bar graph - brute force



Then, the application of a branch-and-bound algorithm is proposed, which is based on the solution found by brute force to eliminate branches from the tree of solutions that have already proved to be more costly. This strategy reduces the search space and accelerates the processing time, maintaining the search for the optimal solution. The efficiency and quality of the solutions obtained by both approaches will be compared, as well as their processing times. From this, it can be seen that the branch and bound algorithm has a solution with better complexity than brute force, with a running time equal to $\mathcal{O}(n^2 * 2^n)$, as shown in the graph below. Observe an exponential increase in the bars due to the number of stores, but also variations depending on the amount of items the truck can carry:

Figure 2. Bar chart - branch and bound



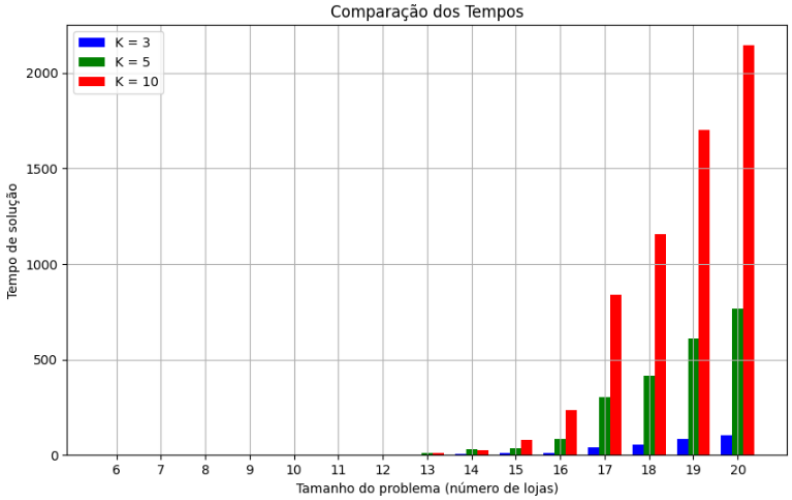
As can be seen in Figure 2, from 16 stores the path calculation starts to consider only K = 3, as it strongly limits the execution time of the program, which starts to accelerate due to its exponential complexity.

Unfortunately, for VRP and NP-type problems, a solution to find the upper or lower bound definitively, quickly, and consistently is unknown. However, it is important to point out that, in practical situations, providing lower or upper limits as a measure of quality is of great relevance for a given proposed solution.

Although the branch-and-bound already has several pruning conditions that result in very efficient approaches, the upper bound or lower bound calculations can be improved by different methods, resulting in faster solutions, although with lower accuracy. Therefore, using the branch-and-bound and changing the limit calculation, as well as adding more restrictions, the alternative branch-and-bound was developed, which aims to have a faster execution, even if it means a decrease in the solution quality.

Several methods for calculating the lower bound were observed: *"Nearest Neighbor Algorithm"*, which calculates the shortest theoretical path based on the shortest distances to the next store; *"Minimum Spanning Tree"*, which uses Kruskal or Prim algorithms to analyze the best alternative; *"Held-Karp Bound"*, which at each iteration classifies a value based on edges and changes the boundary; *"Cheapest Link"*, which calculates the value of a theoretical cycle using the smallest values of each vertex. [Cook], [Urrutia et al. 2007], [Sal], [Rathinam and Sengupta 2006].

Figure 3. Bar chart - branch and bound - alternative



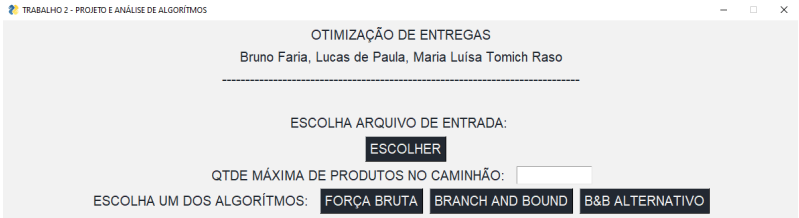
3. Implementation

Firstly, a graphical interface was created that presents the possibility of choosing the input file and the type of solution to be used. Both are further described in the subsections below. In addition, a secondary interface was created, which appears after choosing the solution algorithm, and which presents the graphical result of the stores along the optimized path, traveled by the truck, showing the change of load and fuel between its branches.

3.1. Interfaces and input file model

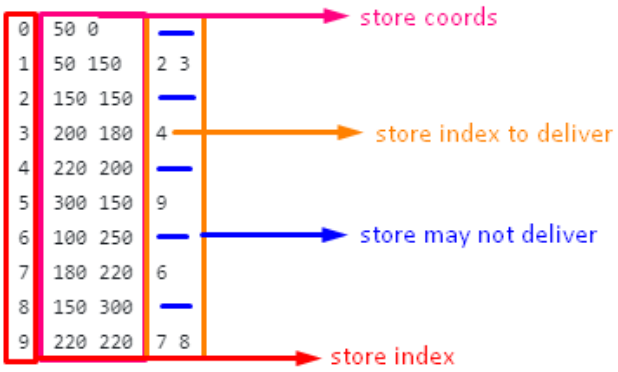
The following interface is shown when the program runs:

Figure 4. Program home interface.



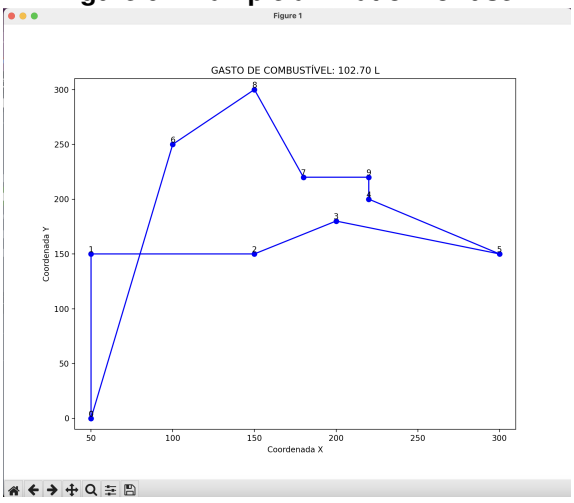
In it, it is possible to choose an input file, the number K of items in the truck, and the desired solution. The input file is in text format, and has the following structure:

Figure 5. Input file structure.



Thus, depending on the choices made in the initial interface, a second interface is dynamically shown, in which a graph is plotted with the stores (with point formats) in their respective coordinates on the map (which was read from the input file). and the optimized path between them is traced according to the chosen approach, also showing the updated fuel consumption and updated truck loads at each point. See, below, an example, remembering that the image only shows the final format of the graph, but the user can see it being built little by little:

Figure 6. Example animation for user



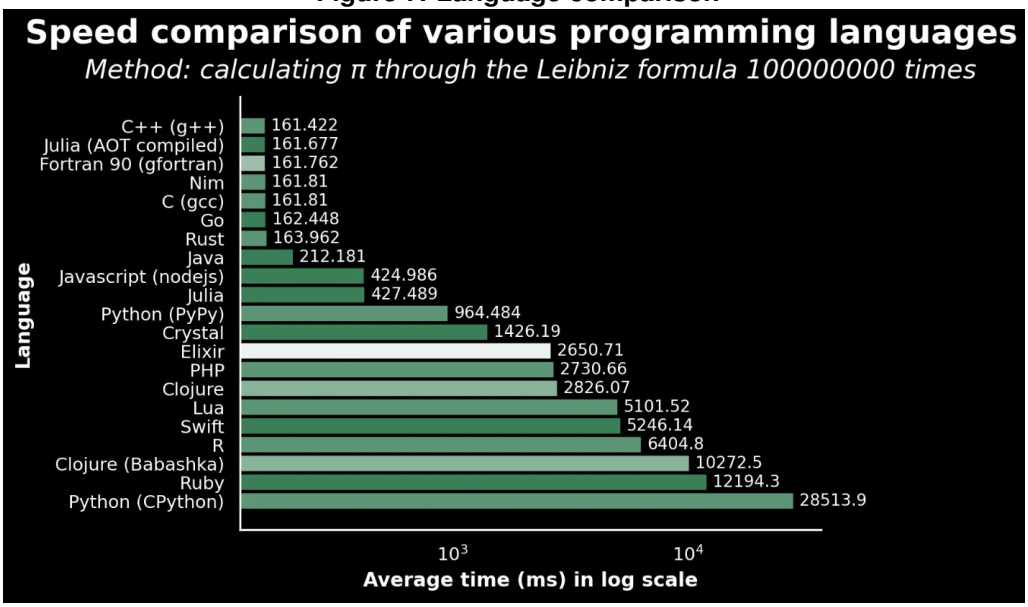
3.2. Choice of programming language

The language chosen for the implementation of the proposed solution was the Python language, for the following reasons:

1. Ease of building interfaces
2. Pivot graph plotting facility
3. Simplicity and efficiency in coding - variables, scope, functions

However, some challenges were also found in its use, such as the execution time compared to other languages. For example, when reaching a certain number of stores, the program starts to take a long time, being unfeasible in brute force and viable in branch and bound, as shown in the language comparison chart:

Figure 7. Language comparison



3.3. Brute Force

The brute force algorithm presented here addresses the problem of finding the optimal delivery route in a logistics scenario with multiple stores and deliveries. This article provides a high-level explanation of the logic behind the algorithm without delving into the details of specific variables.

Given a set of stores, deliveries, a distance matrix, and a constraint on the number of products a truck can transport, the objective is to determine the most efficient route that minimizes the total delivery cost. The algorithm considers both the distance traveled and fuel consumption as factors in determining the solution.

Starting from the set of stores, the brute force algorithm tests all solution possibilities until the end, that is, it does not apply any type of pruning, which is why its complexity is factorial and, therefore, very high. However, he must still assess whether the store is valid, seeing whether all products were delivered and whether it arrived empty at the destination, as there may be a path that is valid with lower fuel costs but that does not meet the standard constraints of the problem.

3.3.1. Brute Force Algorithm Overview

Initialization:

- Initialize a variable to store the number of permutations.
- Extract the store IDs from the input data, excluding source and destination.
- Set best route and initial cost to undefined.
- Initialize variables to store the best cost, the list of costs to plot, the list of carried items, and the permutation count.

Generate permutations:

- Recursively generate all possible permutations of store IDs.

- For each permutation:
  - Add the origin (0) and destination (0) to form a complete route.
  - Increment the permutation count.
  - Call the helper function to calculate the total trip for this permutation.

Calculate Total Travel for a Permutation:

- Initialize variables to store fuel consumption, transported items, and route information.
- Iterate over the stores in the permutation:
  - Check if the store is visitable based on the current route.
  - Whether the store can be visited:
    - Check that the current store is valid based on the transported items and the delivery list.
    - If the branch is valid:
      - Calculate the current fuel consumption.
      - Check if the current cost is better than the previous best cost.
      - If so, update best cost, list of costs to plot, list of transported items and route
- Return the answer to the program.

3.4. Branch and Bound

The branch and bound algorithm presented here addresses the problem of finding the optimal delivery route in a logistics scenario with multiple stores and deliveries. This article provides a high-level explanation of the logic behind the algorithm without delving into the details of specific variables.

Given a set of stores, deliveries, a distance matrix, and a constraint on the number of products a truck can transport, the objective is to determine the most efficient route that minimizes the total delivery cost. The algorithm considers both the distance traveled and fuel consumption as factors in determining the optimal solution.

To arrive at the optimal solution, restrictions must be applied and the fuel cost lower bound calculated in order to prune fruitless branches of the permutation tree, avoiding unnecessary calculations for paths that will not be interesting. Some of the restrictions are: see if the products that have to be delivered have not yet been picked up and the stores have not yet been visited, check if the quantity of products loaded is supported by the truck, see if the current path has a better yield than the old one, with this the branches will be pruned efficiently.

3.4.1. Branch and bound algorithm overview

Initialization:

- Extracts store IDs from input data, excluding source and destination.
- Sets best initial route and cost to undefined..
- Initialize variables to store permutations and number of pruned branches

Generate permutations:

- Recursively generate all possible permutations of store IDs.
- for each permutation
  - Add the origin (0) and destination (0) to form a complete route.
  - Increase permutation count.
  - Call the helper function to calculate the total trip for this permutation.

Calculate Total Travel for a Permutation:

- Initialize variables to store fuel consumption, transported goods and route information.
- Iterate over the stores in the permutation:
  - Check if the store is visitable based on the current route.
  - Whether the store can be visited:
    - Check that the current store is valid based on the transported items and the delivery list.
    - If the branch is valid:
      - Calculate the fuel consumption lower limit for the current branch.
      - If lower bound is better than current best cost: explore branch
      - Otherwise, prune the branch.
    - Otherwise, prune the branch.
  - Otherwise, prune the branch.

Evaluate solution:

- If the current permutation arrives at the destination with all products delivered:
  - Calculate the total cost of the trip.
  - If the cost is better than the previous best cost:
    - Update the best cost and solution route.
    - Store fuel consumption, transported goods, and route information.
- Return the answer to the program.

3.5. Branch and Bound - alternative

The alternative branch and bound algorithm is based on the branch and bound algorithm mentioned above, with the same constraint tests and the same structure, however the observed difference is in the lower bound calculation, which tries to predict the future to execute in a more efficiently, using a heuristic based on *Cheapest Link* and *Nearby Neighbor*. One of the parts of the calculation for the lower bound was the summation:

$$\frac{1}{2} \sum_{v \in V} (sum\ of\ costs\ of\ two\ adjacent\ edges\ a\ v) \tag{1}$$

This equation determines that, for each store 'v', if we consider that two edges pass through it in a path T and add their costs, it is possible to determine that the general sum for all vertices would be twice the cost of the path T, since all edges are considered twice. Furthermore, the observed edges will always be the two smallest edges connected to each 'v' store. Therefore, it is possible to state that this sum comprises "The shortest theoretical path to enter and exit all remaining stores".

3.5.1. Alternative branch and bound algorithm overview

Check the branch:

- Check that the current store is valid based on the items in the truck and the deliveries to be made.
- If the branch is valid:
  - Removes the current store from the temporary stores dictionary.
  - Calculate the sum of the shortest distances from the remaining stores using the distance matrix.
  - Calculate the lower bound estimate by adding the current cost to the sum of the smallest distances.
  - If the lower bound estimate is within 95% of the current best cost:
    - Continue recursively generating permutations with the updated parameters.
  - Otherwise increment the pruning count as the branch is not valid and will be removed.
  - Add the current store to the temporary store dictionary.
- Otherwise increment the pruning count as the branch is not valid and will be removed.

Calculate shortest distances:

- If there are at least three stores left:
  - Initialize a dictionary to store the smallest distances for each store.
  - Iterate over each store:
    - Calculate the distances from the current store to all other remaining stores
    - Sort the distances in ascending order.
    - Keep only the two smallest distances for each store.
    - Store the smallest distances in the dictionary for the current path.
    - Calculate the sum of all smaller distances for later use.
  - Calculate the lower bound estimate by dividing the sum of all smaller distances by 20 (assuming a fuel efficiency of 10).
- Otherwise, if there are two stores left:
  - Calculate the distance between stores.
  - Return the distance divided by 10 (assuming a fuel efficiency of 10) as the lower bound estimate.
- If there is only one store left or no stores left, return 0.0 as the lower bound estimate, as its calculation is no longer needed.

4. Test report

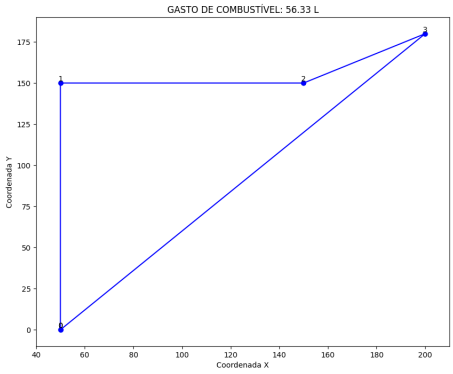
4.1. Test 0: 3 stores

Test 0 input file:

Figure 8.

```
0 50 0
1 50 150 2 3 2
2 150 150
3 200 180
```

Figure 9. Path 3 Stores



4.1.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3,5,10 (equal)	0.0	[0, 1, 2, 3, 0]	56.33

4.1.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.1.3. Brute force

K	TIME (s)	STORES	FUEL (L)
3, 5 and 10	0.0	[0, 1, 2, 3, 0]	56.33

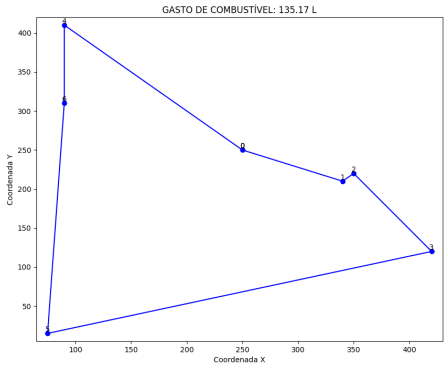
4.2. Test 1: 6 stores

Test 1 input file:

Figure 10.

```
0 250 250
1 340 210 3 4 5
2 350 220 6
3 420 120
4 90 410
5 75 15
6 90 310
```

Figure 11. Path 6 stores



4.2.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.0	[0, 1, 3, 2, 5, 6, 4, 0]	143.90
5, 10	0.0	[0, 1, 2, 3, 5, 6, 4, 0]	135.17

4.2.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.2.3. Brute force

K	TIME (s)	STORES	FUEL (L)
3	0.02	[0, 1, 3, 2, 5, 6, 4, 0]	143.90
5, 10	0.02	[0, 1, 2, 3, 5, 6, 4, 0]	135.17

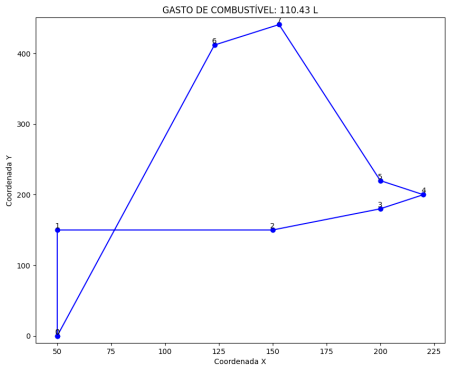
4.3. Test 2: 7 stores

Test 2 input file:

Figure 12.

```
0 50 0
1 50 150 2 3
2 150 150 4 5
3 200 180
4 220 200
5 200 220 6 7
6 123 412
7 153 441
```

Figure 13. Path 7 Stores



4.3.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.0	[0, 1, 2, 3, 4, 5, 7, 6, 0]	110.42
5, 10	0.0	[0, 1, 2, 3, 4, 5, 7, 6, 0]	110.42

4.3.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.3.3. Brute force

K	TIME (s)	STORES	FUEL (L)
3, 5 and 10	0.18	[0, 1, 2, 3, 4, 5, 7, 6, 0]	110.42

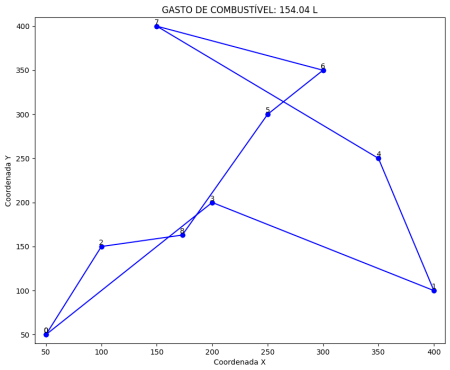
4.4. Test 3: 8 stores

Test 3 input file:

Figure 14.

```
0 50 50
1 400 100 7 2
2 100 150
3 200 200 4
4 350 250
5 250 300 8
6 300 350
7 150 400 6 5
8 173 163
```

Figure 15. Path 8 Stores



4.4.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.03	[0, 3, 1, 4, 7, 6, 5, 8, 2, 0]	154.03
5.10	0.05	[0, 3, 1, 4, 7, 6, 5, 8, 2, 0]	154.03

4.4.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.4.3. Brute force

K	TIME (s)	STORES	FUEL (L)
3, 5 and 10	1.56	[0, 3, 1, 4, 7, 6, 5, 8, 2, 0]	154.03

4.5. Test 4: 9 stores

Test 4 input file:

```
0 50 0
1 50 150 2 3
2 150 150
3 200 180 4
4 220 200
5 300 150 9
6 100 250
7 180 220 6
8 150 300
9 220 220 7 8
```

Figure 16.

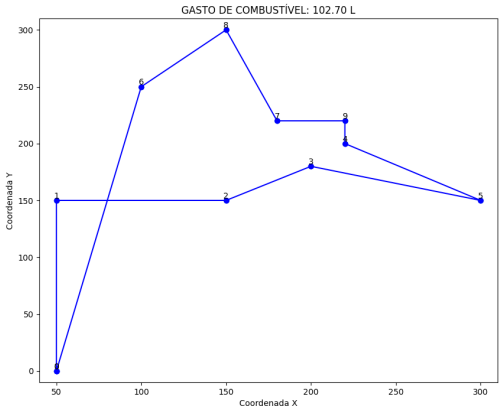


Figure 17. Path 9 Shops

4.5.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.06	[0, 1, 2, 3, 5, 4, 9, 7, 8, 6, 0]	102.70
5, 10	0.13	[0, 1, 2, 3, 5, 4, 9, 7, 8, 6, 0]	102.70

4.5.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.5.3. Brute force

K	TIME (s)	STORES	FUEL (L)
3, 5 and 10	15.86	[0, 1, 2, 3, 5, 4, 9, 7, 8, 6, 0]	102.70

4.6. Test 5: 10 stores

Test 5 input file:

```
0 50 0
1 50 150 2 3
2 150 150
3 200 180 4
4 220 200
5 300 150 9
6 100 250
7 180 220 6
8 150 300 10
9 220 220 7 8
10 421 435
```

Figure 18.

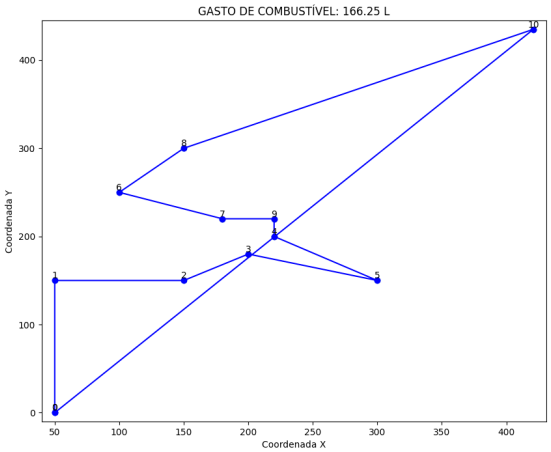


Figure 19. Path 10 Stores

4.6.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.21	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 0]	166.24
5, 10	0.66	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 0]	166.24

4.6.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.6.3. Brute force

K	TIME (s)	STORES	FUEL (L)
3, 5 and 10	175.01	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 0]	166.24

4.7. Test 6: 11 stores

Test 6 input file:

```
0 50 0
1 50 150 2 3
2 150 150
3 200 180 4
4 220 200
5 300 150 9
6 100 250
7 180 220 6
8 150 300 10
9 220 220 7 8
10 421 435 11
11 351 141
```

Figure 20.

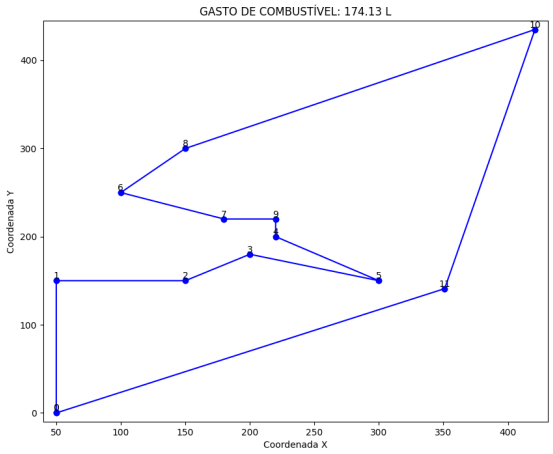


Figure 21. Path 11 Shops

4.7.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.38	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 11, 0]	174.12
5, 10	1.21	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 11, 0]	174.12

4.7.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.7.3. Brute force

At this point, it becomes impossible to execute the brute force algorithm in a viable time, since it has taken on too large proportions.

4.8. Test 7: 12 stores

Test 7 input file:

```
0 50 0
1 50 150 2 3
2 150 150
3 200 180 4
4 220 200
5 300 150 9
6 100 250
7 180 220 6
8 150 300 10
9 220 220 7 8
10 421 435 11 12
11 351 141
12 361 114
```

Figure 22.

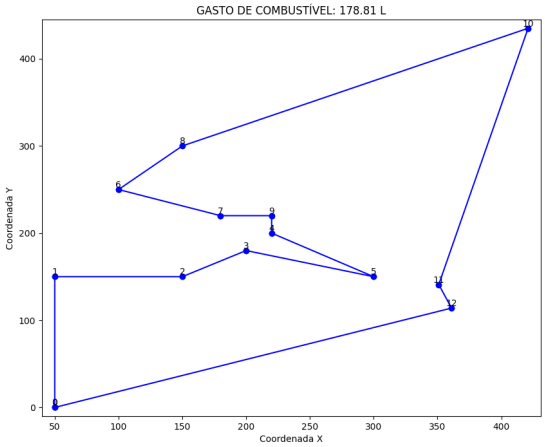


Figure 23. Path 12 Shops

4.8.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	0.63	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 11, 12, 0]	178.81
5, 10	2.23	[0, 1, 2, 3, 5, 4, 9, 7, 6, 8, 10, 11, 12, 0]	178.81

4.8.2. Branch and bound - Alternative

At this time, a negligible difference was observed from the running time of normal Branch and Bound to Branch and Bound using alternate. In this way, it is no longer a relevant solution to the problem.

4.8.3. Brute force

At this point, it becomes impossible to execute the brute force algorithm in a viable time, since it has taken on too large proportions.

4.9. Test 8: 13 stores

Test 8 input file:

```
0 50 50
1 100 500
2 150 350 11
3 200 400
4 250 100 7 6
5 300 150
6 350 200 5 1
7 500 0 13
8 450 250
9 400 300 8
10 0 450 9
11 125 475 3
12 175 325
13 225 375 12
```

Figure 24.

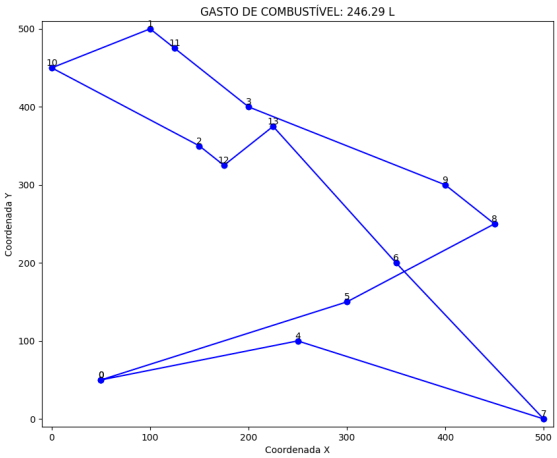


Figure 25. Path 13 Shops

4.9.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	6.37	[0, 4, 7, 6, 5, 13, 12, 2, 10, 1, 11, 3, 9, 8, 0]	249.16
5	35.06	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 3, 9, 8, 5, 0]	246.28
10	36.72	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 3, 9, 8, 5, 0]	246.28

4.9.2. Branch and bound - Alternative

K	TIME (s)	STORES	FUEL (L)
3	2.64	[0, 4, 7, 6, 5, 13, 12, 2, 10, 1, 11, 3, 9, 8, 0]	249.16
5	8.47	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 3, 9, 8, 5, 0]	246.28
10	8.57	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 3, 9, 8, 5, 0]	246.28

The same result is observed for fuel and paths, however with a visible change in execution time.

4.9.3. Brute force

At this point, it becomes impossible to execute the brute force algorithm in a viable time, since it has taken on too large proportions.

4.10. Test 9: 14 stores

Test 9 input file:

```
0 50 50
1 100 500
2 150 350 11
3 200 400
4 250 100 7 6
5 300 150
6 350 200 5 1
7 500 0 13
8 450 250
9 400 300 8
10 0 450 9
11 125 475 3
12 175 325
13 225 375 12 14
14 131 431
```

Figure 26.

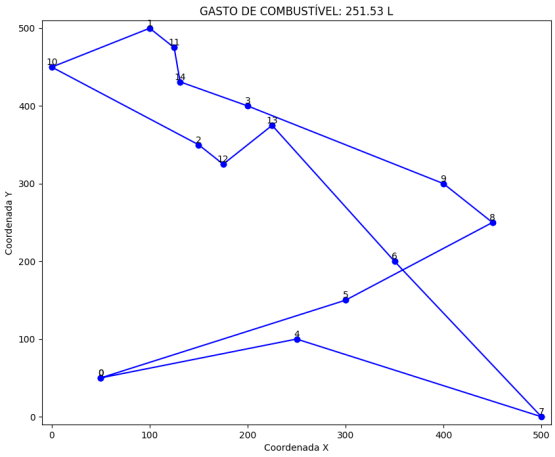


Figure 27. Path 14 Shops

4.10.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	9.81	[0, 4, 7, 6, 5, 13, 12, 2, 14, 11, 1, 10, 3, 9, 8, 0]	254.25
5	117.2	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 14, 3, 9, 8, 5, 0]	251.52
10	129.98	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 14, 3, 9, 8, 5, 0]	251.52

4.10.2. Branch and bound - Alternative

K	TIME (s)	STORES	FUEL (L)
3	5.61	[0, 4, 7, 6, 5, 13, 12, 2, 14, 11, 1, 10, 3, 9, 8, 0]	254.25
5	31.39	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 14, 3, 9, 8, 5, 0]	251.52
10	27.21	[0, 4, 7, 6, 13, 12, 2, 10, 1, 11, 14, 3, 9, 8, 5, 0]	251.52

The same result is observed for fuel and paths, however with a visible change in execution time.

4.10.3. Brute force

At this point, it becomes impossible to execute the brute force algorithm in a viable time, since it has taken on too large proportions.

4.11. Test 10: 15 stores

Test 10 input file:

```
0 400 400
1 320 120
2 189 283
3 176 341 1 2
4 234 472
5 134 164
6 451 361 5
7 153 174 4
8 421 341 9 10 11
9 125 153
10 163 131 12
11 114 261 13
12 123 423
13 165 165 14 15
14 412 235
15 500 324
```

Figure 28.

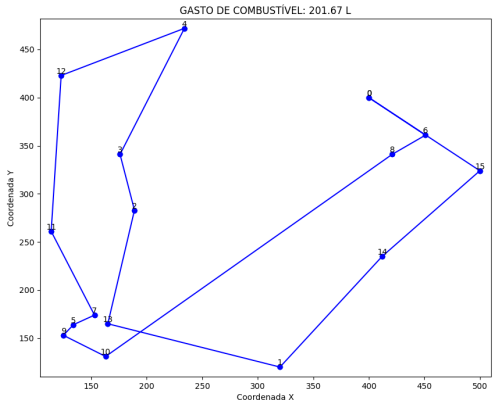


Figure 29. Path 9 Shops

4.11.1. Branch and bound

K	TIME (s)	STORES	FUEL (L)
3	58.08	[0, 8, 10, 9, 11, 12, 3, 2, 13, 1, 14, 15, 6, 7, 5, 4, 0]	257.57
5	540.12	[0, 6, 8, 10, 9, 5, 7, 11, 12, 4, 3, 2, 13, 1, 14, 15, 0]	201.67
10	630.81	[0, 6, 8, 10, 9, 5, 7, 11, 12, 4, 3, 2, 13, 1, 14, 15, 0]	201.67

4.11.2. Branch and bound - Alternative

K	TIME (s)	STORES	FUEL (L)
3	9.39	[0, 8, 10, 9, 11, 12, 3, 2, 13, 1, 14, 15, 6, 7, 5, 4, 0]	257.57
5	34.70	[0, 6, 8, 10, 9, 5, 7, 11, 12, 4, 3, 2, 13, 1, 14, 15, 0]	201.67
10	79.70	[0, 6, 8, 10, 9, 5, 7, 11, 12, 4, 3, 2, 13, 1, 14, 15, 0]	201.67

The same result is observed for fuel and paths, however with a visible change in execution time.

4.11.3. Brute force

At this point, it becomes impossible to execute the brute force algorithm in a viable time, since it has taken on too large proportions.

4.12. MACHINE

In order to provide more detailed information about the experimental environment used, we present in the **Table 1** the specifications of the computer used in the tests carried out in this study. The data contained in this table includes information about the processor, amount of RAM memory, operating system, among others.

Specifications	
PROCESSOR	i5-10400F 2.90GHz
RAM MEMORY	16 GB - 2666mhz - DDR4
SIS. OPERATIONAL	Windows 10
VIDEO CARD	RTX 3060 - 12GB

Table 1. Specification of the machine used for the tests

5. Conclusion

Based on the tests carried out, an interesting behavior was observed: as the number of stores increases, the number K of items that the truck can carry affects the final path and increases the execution time. This relationship occurs due to the existence of more valid paths in the branch and bound algorithm, therefore less pruning occurs, which consequently generates longer execution time. However, this trend is not significant when it comes to small numbers of stores.

It is important to point out some considerations about the functioning of the algorithms. The delivery problem between branches is classified as NP, which means that using brute force is not a viable option when dealing with very large input files, as observed in the tests in the previous section. Processing time becomes unfeasible, as it would be necessary to explore all permutations of all paths. In this scenario, the branch and bound technique appears as an alternative to reduce execution time, as it allows pruning unpromising branches of the tree, saving resources and time that would be wasted with them. From there, it becomes possible to handle larger inputs. In addition to this factor, one can also note the existence of a second branch and bound method, which makes the time even smaller, which consists of adding heuristics to the code to generate more promising pruning, which visibly makes the code faster.

Therefore, it is important to emphasize again that the choice of programming language can also affect small characteristics of the final solution, due to its particularities and specific limitations. Even using a language with 100x better performance (For example C++), the improvement would be linear, thus becoming irrelevant in the next iterations, due to the complexity of the algorithms chosen to deal with the problem, as discussed throughout the document.

6. Bibliography

References

The travelling salesman problem. https://www.thechalkface.net/resources/Travelling_Salesman_England.pdf. Accessed on June 4, 2023.

Cook, W. The traveling salesman problem. https://math.mit.edu/~goemans/18433S15/TSP-CookCPS.pdf. Acessado em 4 de junho de 2023.

Rathinam, S. and Sengupta, R. (2006). Lower and upper bounds for a multiple depot uav routing problem. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 5287–5292.

Urrutia, S., Ribeiro, C. C., and Melo, R. A. (2007). A new lower bound to the traveling tournament problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 15–18.