Optimal Inspection Routes Problem – ASAE Use Case

David Burchakov, *FEUP*, André Dias, *FEUP*

Abstract

Metaheuristic algorithms are computational intelligence paradigms especially used for sophisticated solving optimization problems. This paper focuses on solving an optimization problem using different meta-heuristic algorithms, including *hill-climbing, simulated annealing, tabu search, and genetic algorithm*. The aim is to compare and test different parameterizations of these algorithms in terms of their average quality of solution and the average time taken to obtain the solution. The results are presented using an appropriate visualization in text and graphic mode. A user-friendly graphical user interface was developed to facilitate the comparison of algorithms, enabling the selection of different algorithms and input of various parameters. Python was used along with several libraries including *numpy*, *pandas*, *matplotlib*, *folium*, and *tkinter* to develop this project. The *folium* library was used to create interactive maps. *Tkinter* was used to create a graphical user interface for the project, providing a simple and intuitive way for users to interact with the program. The *prettytable* and *collections* libraries were also used to facilitate data management and analysis. Additionally, the *time* and *functools* libraries were used to aid in performance optimization.

I. Introduction

As information technology continues to advance, an increasing number of optimization problems are arising in various fields, including engineering, economics, finance, logistics, transportation, telecommunications, computer science, physics, etc. Unfortunately, many optimization problems are classified as NP-hard problems, which cannot be solved in polynomial time, unless NP is equal to P. This limitation means that they can't be solved in a reasonable amount of time for large problem sizes. Instead of giving up, researchers thought to use possible workarounds (approximation methods) which can find a good enough (but not exact!) solution in a reasonable time, these methods can be categorized to *heuristics* and *meta-heuristics*. The main difference between the two is that *heuristics* are more problem-dependent than *meta-heuristics*. Heuristics *(from Greek εὑρίσκω "I find, discover")* can efficiently solve a specific problem but may be insufficient for other problems. In contrast, meta-heuristics appear to be a generic algorithm that can be applied to almost all optimization problems. In spite of the prosperity of meta-heuristic in solving a given problem, it can't solve all optimization problems, and all meta-heuristics have the same performance on average.[1]

II. Definition of the problem

The practical application of this work is in solving the "Optimal Inspection Routes Problem" of the Portuguese Economic and Food Safety Authority (ASAE). The problem is to plan and allocate resources to cover inspections in the most optimal way. The paper provides a simplified version of the problem and uses a real-world dataset containing 1000 establishments in the Porto district, divided into two CSV files. The 'establishments.csv' information includes ID, name, latitude, longitude, inspection duration, inspection utility, and opening hours. The second file, 'distances.csv', contains the time distances between every one of these establishments. The problem instance is an instance of the *Vehicle Routing Problem*. and the goal is to inspect all the establishments in the minimum possible time, considering travel, waiting, and inspection time. Different constraint combinations are presented in multiple scenarios, and the paper explores the possibilities of different constraint combinations.

A. Travel time minimization

In this scenario there are finite number of available vehicles – k**.** K can be defined as *floor(0.1 x n)* where *n* is the dataset size. So, for instance, given 200 establishments, 20 vehicles must be used. Their number of working hours is unlimited. The objective is to minimize the time required to inspect all establishments, taking into account travel, waiting, and inspection time.

B. Resource allocation minimization

In this variant there are an infinite number of available vehicles. Each ones' route must not exceed the 8 working hours by any chance. The vehicles are not required to finish their routes in the departure point, i.e. the route is considered to be finished immediately after the last inspection. The goal in this problem is to inspect all the establishments using the minimum number of vehicles possible. The inspection's duration must be considered 5 minutes.

III. Formulation of the problem as an optimization problem

Now that we have defined the problem, for both scenarios, we must formulate it in order to treat it like an optimization problem. The problem state space is the same for both problems since the solutions (as we are going to see ahead) are a recombination of establishments. So, considering *n*

[1] This paper was submitted on the 20th of March 2023. This work was done as a final project of the curricular unit of Artificial Intelligence, lectured in the second semester at the Faculty of Engineering of the University of Porto (FEUP), as a part of the Master's in Engineering and Data Science.

André Dias is a student of the Master's in Mechanical Engineering, at FEUP (e-mail: up201904568@fe.up.pt).

David Burchakov is a student of the Bachelor in Informatic Engineering, at FEUP (e-mail: up202203777@fe.up.pt).

establishments, the state space is *n!*. If we consider the whole data set of 1000 establishments, the state space reaches an extraordinary number of 4^{2567} (considering the real-world problem, with almost 3 million establishments registered, the state space can be considered approximately infinite). Each scenario has its own solution representation.

A. Solution representation

1) First scenario

Keeping in mind that in the first scenario each vehicle has to start in the Depot (represented by the ID 0) and also end the inspections there, we must consider that aspect in the solution representation. As the problem defines *0.1*n* vehicles for *n* establishments, we can consider each vehicle always inspects 10 economic operators. Therefore, the solution representation will be an array with *0.1*n* routes, each route corresponding to a vehicle and being itself an array of 12 establishments (10 plus 2 times the Depot). For example, considering a data set of 20 establishments and, consequently, 2 vehicles, the solution would be {Route1, Route2} and Route1 would be, for example: {0 20 11 10 3 6 1 2 9 17 12 0}. In this way, to achieve different solutions all we have to do is lock the zeros in their places and then permutate the rest of the numbers.

2) Second scenario

The big difference in the second scenario is we don't know how many vehicles are going to be inspected for each vehicle, since that is what we want to minimize. Therefore, we cannot add the zeros in specific locations like in the first case. Keeping that in mind, all we are going to have in the solution representation is really an array with a combination of the *n* establishments. We can also interpret this similarly to the previous case, if we consider the solution is an array of routes, with each route having a different number of establishments, depending on the fulfillment of the 8 hours of maximum work.

A very important consequence of this approach is that the evaluation function for this problem can't forget to include the distance between the Depot and the first establishment when counting time, since they all start their routes there (even though they now finish in the last operator, and not on the Depot).

B. Neighborhood, mutation and crossover operators

Most optimization algorithms use local search to explore a certain region of the state space and find the optimal solution, so it is necessary to create a neighborhood function. From all the algorithms implemented (detailed ahead), Genetic Algorithms are the only ones that are not based on the idea of local search (although it can be integrated, resulting in the so-called Memetic Algorithms), so we also need to characterize its mutation and crossover functions.

Neighborhood and mutation operators are typically applied in different ways, but have the same purpose, which is to find solutions that are similar to the one we have, in order to better explore the search space. The big difference is that neighborhood functions are used in local search, which implies that it is intelligent, meaning it will search for a better near solution. Mutation is used in genetic algorithms and is based in the natural evolution of species, as we are going to see ahead. Therefore, like in nature, the mutation is typically random. In this case, mutation and neighborhood functions are going to be random or specific permutations in the solution's establishments, respectively.

Crossover is also used in genetic algorithms, as a recombination of two solutions (parents). In this case, a variant of the Travel Salesman Problem, crossover has to be done ordered-based, since we cannot have the same establishments more than once in the solution, and what we are really looking for is the order in which they appear. The operator used is going to be analyzed further in this report.

C. Constraints and evaluation functions

In both cases there is not going to exist constraints, since all the restrictions are assured either by the solution representation or by the evaluation function. All the requirements were already described in the problem description. For example, the way the solution is represented and the operators (mutation, crossover, etc.) are applied, already assures there are no establishments inspected twice.

The evaluation function is made with the purpose of attributing a score to each solution, evaluating each one according to the specifications of the problem. In the first case lesser the total time to inspect all the operators, highest the score, whereas in the second problem the score will be higher for individuals that use a smaller number of cars. The practical implementation for each scenario will be further described.

IV. Implementation details

The programming language used in this practical work was Python. As it was developed by the two authors of this report, the development environment chosen in order to share the code was Google Colab.

The input data, as it is an excel file with all the needed information, must be imported into our program in order for it to work. We allowed this process to be done in two ways. One is trough google drive, which allowed us to work with the same code in different computers. The other option, implemented as a plan B, is to import the data locally, from the computer files.

We used object-orientated programming to define the problem and the evaluation functions. The algorithms implemented are a group of functions, each one developed for a specific function, within the algorithm. This work method allowed us to make generic algorithms, that could be adapted for each scenario of the problem.

Several python libraries were imported to allow a better use of the data, a more efficient program and better results display.

V. Algorithms implemented

A. Hill Climbing

The most basic technique used in this work was the Hill Climbing, which is a local search optimization algorithm. Its implementation was mostly to have a first method to compare with the other, more efficient, algorithms.

The main idea behind hill-climbing is to iteratively improve a solution by making small changes, which will either increase or decrease its evaluation function value. The name comes from the fact that it metaphorically climbs a hill until reaching the best solution (or the highest point). The main problem is it gets stuck in local optima, so in practice it is only used to

solve really simple problems or like in this case, to act as a comparison method.

B. *Genetic algorithm*

In 1858, Charles Darwin and Alfred Russel Wallace independently proposed the theory of evolution by natural selection (although it is commonly associated to Darwin).

The modern theory of evolution combines genetics and Darwin and Wallace's ideas of natural selection, creating the basic principle of Population Genetics, which states that "variability among individuals in a population of sexually reproducing organisms is produced by mutation and genetic recombination". In 1975, Holland presented the Genetic Algorithm, adapting the biological evolution to solve optimization problems in real-world scenarios.

The algorithm is a stochastic and evolutionary process, that starts with a random population, consisting in possible solutions to the problem, and evolves until finding an optimal individual.

In this case, we started with a random permutation of the establishments, representing each individual as described in both solutions representations, giving us random routes. Each individual was assigned a score according to the fitness of the solution, similar to assessing how effective an organism is when competing for available resources, in nature. For this purpose, we used the evaluation (or fitness) functions.

In each generation, we used the selection operators to select some individuals to be parents, and then applied the crossover operator to generate the offspring. Then mutation was applied to the offspring, and the resulting individuals were inserted into the population, replacing a percentage of the worst individuals.

We also introduced a very small probability of using the neighborhood operator instead of the mutation operator, performing local search in very few individuals, instead of random permutations. This method can sometimes find better solutions, exploring the neighborhood of certain good individuals. It contributes to intensification, which means, accelerating convergence, but also diversification, allowing to better explore the search space. However, it also increases computational effort.[1]

This iterative process continued generation after generation, until the stopping criteria would be satisfied, which was having a certain number of generations without change in the best fitted individual.

C. *Simulated Annealing*

This algorithm is a very well-known metaheuristic, inspired by the thermodynamical process present in the cooling of a material in a heat bath. Its name comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

The function to be minimized is analogous to the internal energy of the system. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.

[1] This process turned our algorithm into a very simple Memetic Algorithm, which is a fusion of population-based search and heuristic learning (local search). It is an adaptation of the Genetic Algorithm and considers the cultural transmission in the genetic evolution, which means an individual, rather than only evolving, can actually learn during his life and change his fitness.

It chooses a random neighbor. The thing that distinguishes it from local search is that it can escape local optima, by accepting all the improving moves, but also some deteriorating moves, that are accepted depending on the amount of deterioration and the temperature (a parameter that decreases with time, similar to the analogy this algorithm is based on). This means that when we are reaching a solution, we have a lower probability of accepting "bad" solutions.

To perform the local search, we used two approaches: applying the neighborhood operator and using a random search to find the neighbor, based on random hill-climbing. To assess the quality of the solution the evaluation functions of each variant of the problem were used. We used different cooling rates for the temperature decrease and the stopping criteria was either a certain number of iterations or a number of iterations without change in the solution. The probability of accepting deteriorating moves was determined by $e^{\frac{-\Delta}{t}}$, being Δ the difference between the new score and the previous score and t the current temperature.

D. *Tabu Search*

Tabu Search is an optimization algorithm that draws its name from the Tongan word "Tabu", meaning sacred and untouchable. The term was originally used by Polynesian aborigines to indicate things that should not be touched. Created in 1986 by Fred Glover, Tabu Search is based on the principle of short-term memory, temporarily blocking the search from certain areas and directing it towards others using bonuses and penalties.

Similarly to Simulated Annealing, Tabu Search also utilizes local search and permits non-improving moves to exit local optima. By implementing a tabu list that prohibits revisiting previously explored solutions, the algorithm is able to diversify the search and prevent looping.

VI. THE APPROACH

A. *Neighborhood operator*

The operator used to perform the neighborhood exploration, i.e., the local search, was the well-known 2-opt. As any local search operator, it is an iterative process, and starts in an initial solution and continuously looks for improvement opportunities in the neighborhood of that solution.

It works as follows: takes two connections between establishments and reconnects them. If it results in a better score, then the current route is updated.

B. *Crossover operator*

As referred, crossover is applied in genetic algorithms to some solutions selected as parents, to originate their offspring by recombination, inserted into the new population of solutions.

For the reasons already mentioned, the operator used was the two-random-point ordered-based crossover. Basically, takes two parents to generate two children. It generates two random numbers in the range of the solution, and takes the numbers (ID of the establishments) of the first parent to the first child, in the same positions they appear. Then it fills the remaining numbers by the order they appear in the second

parent. It does the inverse process to generate the second child.

C. Mutation operator

As already referred, mutation is a random process, that originates a new solution, based on the existing one. It is performed in genetic algorithms, applied in the offspring that came from crossover, in order to assure a better exploration of the state space when generating a new population. The probability of mutation is the probability that each individual has of being subject to the random operator. It is a crucial parameter in genetic algorithms, once it can't be too high because it would cause too random solutions, which means, slow the convergence rate. However, if it is too low, in some problems, the exploration of all the state space is not assured and the solution gets stuck in local optima. It must be adapted according to the problem and the operator used.

In this case, the mutation operator actuates in the solution by directly permutate two establishments. Considering the size of the data set is variable, and sometimes it can get to a very big number, the number of permutations has to be adjusted in order for it to have impact for all data sizes.

D. Selection operator

To perform the selection of individuals to be parents it was used a combination of the selection via tournament and via roulette.

Tournament is an elitist process, putting into competition a certain number of individuals and choosing the better one. Roulette gives each individual a certain probability of selection, according to its score (or fitness value), so the best individuals have a higher probability of selection. Then it generates a random number several times, to choose the selected individuals, just like if we were spinning a roulette.

In both cases the selection was made to a pool, which will intuitively have more good individuals, since they can appear several times in the pool. From that pool, all candidates have the same probability of being selected as parents.

The size of the tournament can be adjusted, as well as the percentage of individuals in the pool that were selected via tournament and the size of the pool.

E. Evaluation functions

The evaluation function is a critical part of every optimization algorithm. If badly designed or placed, it can have a huge effect on the results and computational effort. Both scenarios used the imported data to calculate time but used different approaches.

As both cases require a lot of computational effort when performing the evaluations, they have to be placed in a way that reduces the amount of times the function is called. For example, in genetic algorithms the evaluation function is called several times for each generation in order to execute the several operators. Placing the evaluation function before the population evolution and converting the scores into a matrix, which could then be used, was one of the techniques used that reduced considerably the computing time, calling the evaluation function only once per generation.

1) First scenario

In this case, the goal was to return a score, based on the total time consumed to inspect all establishments. To do that it counts the duration of work of a vehicle, until inspects 10 establishments and arrives at Depot. It considers the schedule of each establishment, it's inspection duration and the total travel time. To be able to check for availability of the economic operators, it is necessary to compute the time of the day for each vehicle, during its journey. When each vehicle arrives at a certain establishment it has to check if it is open or not. If not, needs to wait until it is, and count the waiting time. Each brigade will have its own total time of work, so the time we took to inspect all the establishments will be the highest of the total times, since they all start at the same hour (9AM).

In the end it returns the negative of the total time since we want to minimize it.

2) Second scenario

In this variant of the problem, the score returned must be based on the number of vehicles required to inspect all the establishments.

This time, the number of establishments inspected varies for each vehicle, as well as the number of vehicles, since we only have to respect the working hours limit. Therefore, the fitness function counts the duration of work of a vehicle, until 8 hours, using the same method to calculate time as the previous case. In this scenario the inspecting time is the same for every establishment, which is 5 minutes.

When it reaches the time limit it counts one vehicle. Then returns the negative of the number of vehicles, as we want to minimize it too.

VII. Experimental Results

The optimization algorithms referred were applied to both variants of the problem (except Tabu Search that was only applied to resource allocation minimization). Next, we present some of the results using different parameters for each scenario. Finally, the results are analyzed and compared, in order to take some conclusions.

A. Hill Climbing

In Fig. 1 we can see the type of output our code provided for hill climbing. It is an example of the travel minimization problem, solved for 20 establishments, using 200 iterations. The graph allows to see the evolution of the best solution through the generations.

As expected, hill climbing was able to find better solutions for all data sizes. However, specially when increasing the data size, it was very clear that it would always get stuck in local optima at some point. The running time of the algorithm was always relatively low and the quality of the solutions varied a lot, depending on the initial random solution. As referred it is a good starting point to compare to the other algorithms.

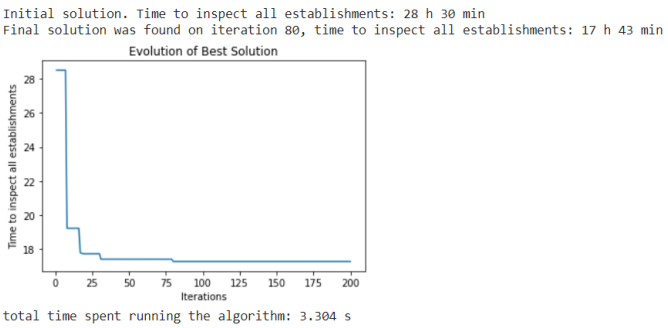

Fig. 1. Output for travel minimization problem, using hill climbing with 20 establishments and 200 iterations.

B. Genetic algorithm

In Fig. 2 it is shown an example of a solution using the Genetic Algorithm. The type of output is similar to the one in Fig. 1, but it also gives a list of routes, one for each vehicle (Fig. 3).

Even though the graph seems to still be decreasing when the algorithm stops, we have to consider that it only represents the solutions until the last improved generation, in this case 45. Actually, if all generations were represented (95), we would see a straight line in the end, meaning the algorithm is finished.

```
Initial solution: Time to inspect all establishments: 27 h 28 min
  Final solution: Time to inspect all establishments: 12 h 1 min
  Found on generation 45
  Number of vehicles utilized: 10
  Number of inspected economic operators: 100
```

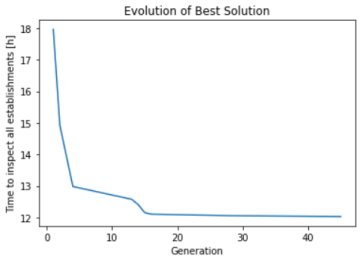

Fig. 2. Output for travel minimization problem, using genetic algorithm with 100 establishments, 50 iterations without change in the best solution and a population size of 30 individuals.

```
+------------+-------------------------------------------------+
|  Vehicle   |                      IDs                        |
+------------+-------------------------------------------------+
| Vehicle 1  |       [ 0 81 21 32 28 48 30 35 46 87 50  0]       |
| Vehicle 2  |       [ 0 39 24 63 68 29 34 18 91 62 95  0]       |
| Vehicle 3  |       [ 0 36 89  5 44 69 93  8 47 41 25  0]       |
| Vehicle 4  |       [ 0 90 55 57 76 13 90 83 37 75 73  0]       |
| Vehicle 5  | [  0  56  49   2   6  66   7 100  43  96  80   0] |
| Vehicle 6  |       [ 0 88 26 51 64 87 41 67 70 73 65  0]       |
| Vehicle 7  |       [ 0 98 24 77 19 40 84 27 79 61  3  0]       |
| Vehicle 8  |       [ 0 97 23 52 17 10  4 92 26 88 45  0]       |
| Vehicle 9  |       [ 0 86 25  8 85 94 33 14 53  1 22  0]       |
| Vehicle 10 |       [ 0 96 58 72 15 99 60 71 89 48 33  0]       |
+------------+-------------------------------------------------+
total time spent running the algorithm: 53.356 s
number of iterations to reach the optimal solution: 45
total number of iterations executed: 95
```

Fig. 3. Complementary output for the same case, with a list of routes, each starting and ending in Depot.

The Genetic Algorithm gave a very good solution to both problems, for every data size (in this case a time reduction of more than 50% of the initial solution). Different parameters resulted in different results, as analyzed in Table I and Table II. The parameters used in the discussion are the ones that gave a better solution.

For big data sizes, the algorithm was usually a little bit slow, comparing to other algorithms. Also, it is important to point out the fact that the quality of the solution also has some dependency on the initial solution, which is random. Keeping that in mind, Table I is a wrap of 10 solutions, using different parameters and different data sizes. It is only a way of getting a comparison, but, in reality, there were many more parameters used and analyzed, even though it is impossible to assure that there wasn't any better solution, since there is an infinite number of options.

Examples	NI	POPS	NE	PC	PS	PT	TS	PM	PLS	Solution
Solution 1	100	20	20	0,7	100	0,4	4	0,5	0,2	15h03min
Solution 2	50	30	20	0,6	80	0,5	4	0,5	0,2	15h46min
Solution 3	50	30	20	0,6	80	0,5	4	0,4	0,2	15h26min
Solution 4	50	30	100	0,7	120	0,5	10	0,7	0,4	12h01min
Solution 5	50	30	100	0,7	120	0,5	10	0,5	0,4	13h28min
Solution 6	50	20	100	0,6	120	0,5	10	0,7	0,3	11h39min
Solution 7	50	20	500	0,7	300	0,5	10	0,6	0,1	13h21min
Solution 8	50	25	500	0,6	300	0,5	14	0,5	0,1	14h23min
Solution 9	50	25	1000	0,6	700	0,4	20	0,3	0,1	15h26min
Solution 10	50	20	1000	0,7	700	0,4	20	0,4	0,05	14h26min

Table I. Result comparison using different algorithms for GA, solving the travel time minimization problem. NI- number of iterations without change in the best solution; POPS- population size; NE- number of establishments; PC- Percentage of the population replaced with the offspring; PS- size of the pool; PT- percentage of the pool selected via tournament; TS- size of the tournament; PM- probability of mutation; PLS- probability of the mutation being replaced by local search.

For the ranges of the population size used, the results weren't much affected by its change. Especially for big data sizes the probability of performing local search was lower, since it increased a lot the computation time.

The probability of performing mutation is a very critical parameter, for the reasons explained above. For example, comparing the solutions 4,5 and 6 in Table I, the best solutions were obtained for higher probabilities. Since the mutation operator was not very intense, meaning that it would perform few changes in the solution, a low probability would result in bad exploration of the search space.

C. Simulated Annealing

In *Fig. 4* it is shown an example of the output of Simulated Annealing, applied to the travel time minimization scenario. Generally the results obtained were good, except for big data sizes in the first scenario, where it converged too fast. In the second scenario, it performed well for all data sizes.

In addition to the same output given by the *SA*, we also developed a visualization in a real-world map (*Fig. 5a, Fig 5b*), using the coordinates of the establishments in the city of Porto. When looking at the results, the visualization sometimes can be less intuitive, but it is important to keep in mind that the routes have to consider the schedules of the establishments, not only the travel time. Therefore, the routes won't be the shortest, but the ones that allow us to get a better solution.

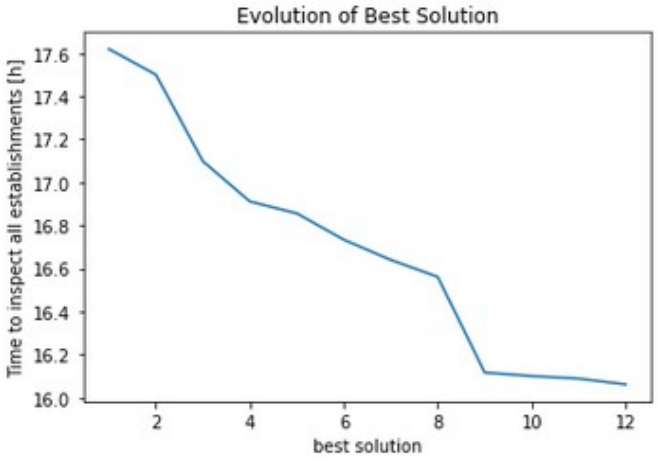

Figure 4: Evolution of the best solution of Simulated annealing algorithm for travel time minimization scenario.

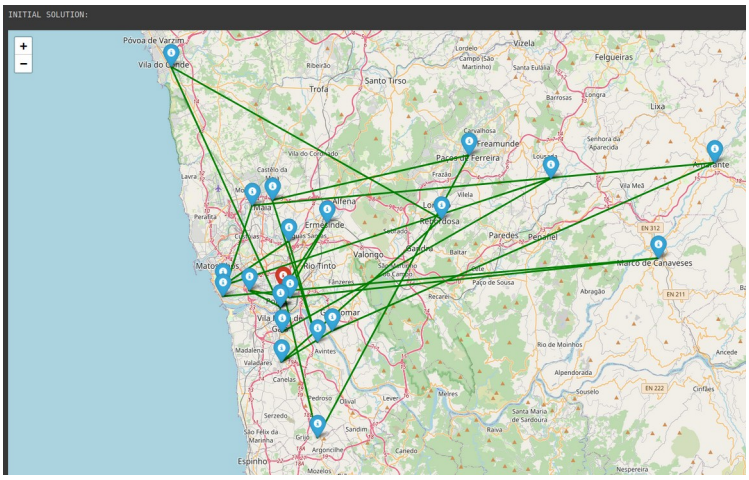

Figure 5a: Initial solution of Travel Time Minimization problem scenario

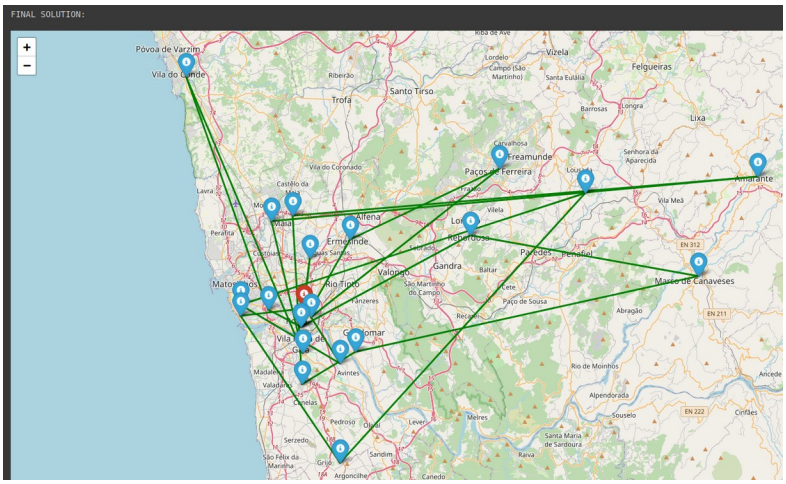

Figure 5b: Final solution of Travel Time Minimization problem scenario

D. Tabu Search

Figure 6 displays the graph of the best solution per generation evolution for Tabu search, which solves the resource allocation minimization problem for 200 establishments. Despite the use of a modest number of establishments, it is evident that the algorithm's run time is long. Nevertheless, the graphical representation allows us to conclude that the Tabu search algorithm achieves an optimal solution in a few iterations.

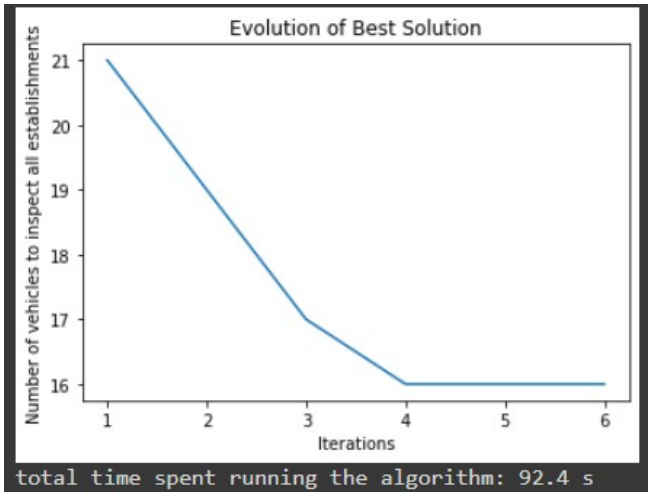

Figure 6: Evolution of the best solution solved by Tabu Search.

VIII. Comparison

A. Hill Climbing vs. Simulated Annealing

Hill climbing and simulated annealing are both optimization algorithms that search for the optimal solution to a problem by iteratively moving towards a higher point (maximum) or a lower point (minimum) in a function landscape.
We tested the algorithms with a large dataset (1000) and many iterations (1000). Both Simulated annealing and Hill climbing algorithms take a negligible amount of time to run (7 and 6 seconds respectfully). The results of running these algorithms are *nearly equal (Figure 7).* Although it is worth noting that Hill climbing is more likely to get stuck in a local optimum and fail to reach the global optimum.
On the other hand, Simulated annealing is more likely to reach the global optimum. Simulated Annealing allows non-improving moves in the search, which enables the algorithm to escape from local optima.

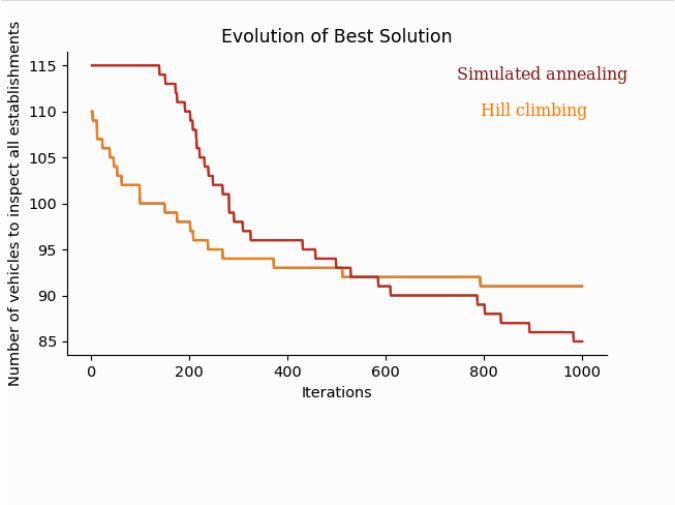

Figure 7: Evolution of the best solution Simulated annealing vs. Hill climbing

B. Genetic Algorithm vs. Tabu Search

Genetic algorithm is a global search algorithm, whereas Tabu search is a local search algorithm that is designed to handle complex problems with many local optima. Both GA and TS require considerable computational power to solve problems with large datasets. For that reason, a humble dataset of 100 establishments and 25 iterations was decided to use.
For TS: tabu list size of 5, tabu tenure of 3.
For GA: population size of 25.
Genetic algorithm seems to require a larger number of iterations to reach a satisfactory result, while Tabu search, though taking a long time to run one iteration, does not require many iterations to get an optimal solution.
Overall, Genetic algorithm requires a larger population size and more computational resources, while Tabu search can be more computationally efficient.

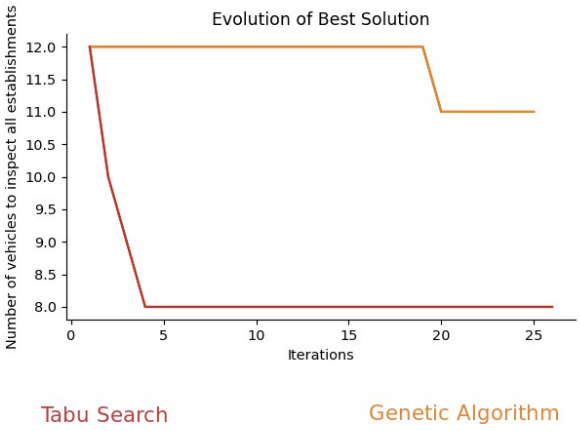

Figure 8: Evolution of the best solution Tabu Search vs. Genetic algorithm.

IX. User Interface

During Project development A graphical user interface was designed to simplify the comparison of algorithms. It allows the user to select from different algorithms and enter various parameters.

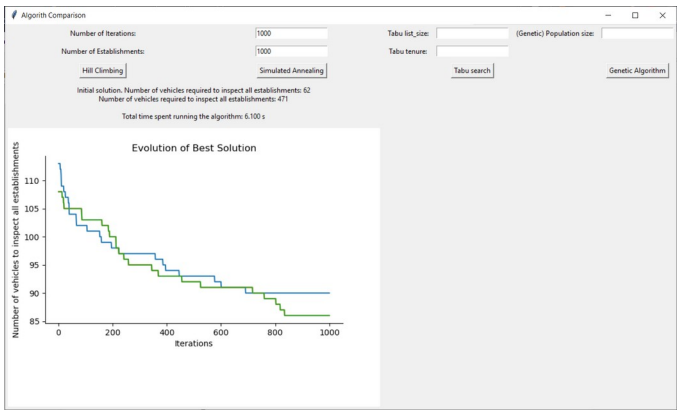

Figure 9: User interface.

X. Conclusion

The most obvious question to conclude this report would be: “What is the best algorithm?” The answer is quite simple: “Depends”. The first and immediate conclusion when discussing optimization algorithms is that the performance of the methods depends on several aspects.

The specifications of the problem, as well as its complexity, have a considerable effect on the quality of the solutions provided by each algorithm (and of course depend a lot on the quality of the code). Therefore, we can only analyze the performance of the algorithms to this specific problem.

Secondly, as we saw in the analysis of the results, the parameters used in each algorithm are crucial to obtain a good approximation of the global optima.

As a final observation, we would like to emphasize a very important aspect. When analyzing a code, we tend to define its quality by the computing time it takes to run the program. However, in optimization problems the reality is that the quality of the solution is much more important. Considering almost all optimization problems in the real-world are a matter of minimizing resources or maximizing results, which means saving or collecting money, every CEO would say that he rather wastes more time if in the end it means getting better results. So, as long as the running time isn’t exceptionally long, the number one criterion to evaluate the quality of an optimization algorithm should be the quality of the solution.

References

[1] Carlos C. A., “Genetic Algorithms”, in *Optimization of Mechanical Systems,* 1st ed., Portugal: Efeitos Gráficos, 2021, pp. 55-100.
[2] Luís P. Reis, “Optimization and Local Search”, in *Artificial Intelligence Lecture 3a*, FEUP, 2022
[3] Luís P. Reis, “Meta-Heuristics - Simulated Annealing and Tabu Search”, in *Artificial Intelligence Lecture 3b*, FEUP, 2022
[4] Luís P. Reis, “Optimization and Genetic Algorithms”, in *Artificial Intelligence Lecture 3c*, FEUP, 2022