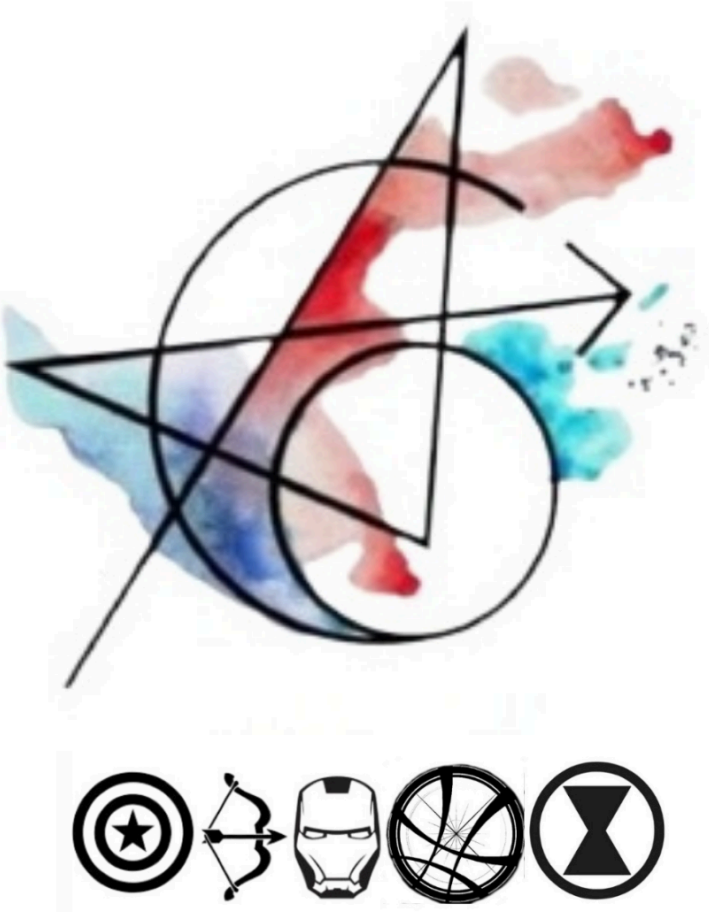
COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION PROJECT
LEONARDO VANNESCHI, BERFIN SAKALLIOGLU
JUNE, 2024
NOVA INFORMATION MANAGEMENT SCHOOL

VEHICLE ROUTING PROBLEM (VRP)



Group Avengers
Devora Cavaleiro, 20230974
Carlos Rodrigues, 20230543
Burcu Yesilyurt, 20230763
David Guarin, 20230602
Lia Margarita Garcia Henao, 20230600

Repository: https://github.com/burcuyesilyurt/vrp_cifo

Index

Statement of contribution: All members contributed equally to the elaboration of this project, despite each person playing a main role on a specific part, all members supported each other on their tasks.

1. Introduction

The selected optimization problem addressed in this report is the Vehicle Routing Problem (VRP) with Time Windows. The aim is to determine an optimal route while considering factors such as the distance, vehicle capacity, and time constraints like pickup/delivery time window start/end. Additionally, as this case is focused on electric vehicles (EVs), the vehicle battery and the position of the charging station are also considered in this problem. In other words, the main goal of this project is to minimize the total distance traveled by all electrical vehicles and reduce the number of vehicles used for deliveries, while considering some constraints. We will address this issue by using different combinations of genetic algorithms (GA), and then by comparing the fitness of each combination we will decide the best solution to this problem.

2. Dataset

The project is based on the "*goeke 2018"* dataset. This dataset contains all the information needed to develop the project, which includes the stations, the pickup, and delivery station, the constraints that we like, vehicle capacity, vehicle charger etc. For further detail on this dataset, you can see in the appendix, on page 8.

2.1. Representation

To represent the individuals (i.e. different routes) of the problem, GA is performed using a list-of-lists approach, where each list represents the route of each vehicle visiting the pickup and delivery locations, starting at, and returning to the depot. Despite the depot position not appearing on the representation, it is considered in the fitness function, as well as the charging stations. As a possible representation, we can consider the following simple example of 3 customers, divided into 3 vehicles, which can do this possible route [[6, 4], [1, 3], [2, 5]]

Each location is represented by the index it appears in our data. Following the same approach used to solve TSP during the classes.

The representation is initiated at random, to avoid introducing bias into the algorithm, but the fitness function ensures that all the requirements are present on the selected individuals.

Apart from this method, we've tried to implement a representation with one list. However, there was a difference between representation length, caused by the capacity to deliver the packages on different routes, which then was filled with dummy variables. Generating problems during developments of both the crossover and mutation operations. Therefore, we decided on our first approach, a list to list.

2.2. Crossover

The following crossover operations were developed and tested in the project, with the probability set to 1 (always do crossover):

- **Single-point Crossover:** In this crossover, we choose a random point from the parents and swap everything after this point between the parents to produce offspring.

- **Cycle Crossover:** Offsprings are generated by selecting elements from the parents and considering cycles to avoid repetitions.
- **Partially mapped Crossover (PMX):** Map parts between two points from one parent to the other and fill in the rest without duplicates.
- **Sequential Constructive Crossover (XO):** Offspring are generated by iterating over the parents, getting one point from a random parent at each iteration.

During the development of the crossover operations, we faced some challenges regarding the constraints of the project. First, with the purpose of implementing the crossover operators we had to flat the list-to-list array to just an array. Then we proceed to do the crossover and generate it again into a list-to-list format. After some attempts we discovered that the algorithm was prone to generate unfeasible solutions, primarily caused by the wrong order of pickup/delivery.

We tried to solve this issue by penalizing infeasible solutions in our fitness function to avoid such scenarios, but even with the penalization, the algorithm continued to generate unfeasible solutions, not converging to a feasible one even though we were using a high number of generations.

To solve this, we've developed a "repair" step that runs after the crossover operations and its goal is to ensure that all routes are feasible.

The repair algorithm consists of the following steps: First, it checks if there's a delivery being done before the pickup and if so, puts the delivery right after the pickup. Then it removes duplicated locations and then adds the missing pickup location right before the delivery. If there is a missing delivery add it right after the pickup. Finally, it adds missing pickup and delivery randomly to a route.

For evaluation purposes, we also developed another repair algorithm very similar to the one described above, with the difference that the locations are placed randomly within the route instead of "right before" or "right after".

This step made the development of crossover operations much easier as we didn't need to care much about violating constraints in the crossover itself and allowed us to use all kinds of crossovers, even position-based crossovers like Single Point Crossover.

The group is aware that this repair step can introduce some modifications that aren't inherited from any parent, but we move forward to it as experimentation of how GA will behave considering it.

2.3. Mutation

The mutation rate in this problem has been defined as 0.15, it guarantees variability across generations, by creating small changes. In this project, we used two mutation operators:

- **Swap:** In this operator, two customers are randomly selected on a route and swapped within the vehicle. This change can lead to new routes, by altering the sequence of the deliveries.
- **Inversion:** This operator simply selects a part of the route and reverses the order of the customers within this part, consequently, this can lead to new possibilities for optimization.

2.4. Selection

For this project, we used Tournament Selection and Fitness Proportionate Selection (FPS):

- **Tournament Selection:** Individuals are randomly selected and one with the best fitness is selected as the parent. Repeat the process to select a second parent.
- **Fitness Proportionate Selection (FPS):** Also known as the roulette wheel. It selects individuals from the population with probabilities proportional to their fitness values.

2.5. Fitness Function

Our process in selecting the fitness function was to first understand the problem, its constraints, and the importance of each variable. Because our main objective is to reduce the amount of time taken by each car, we initially considered the time taken in each route as the most important variable for calculating fitness.

Then, to calculate the total amount of time, we first measured the distance between one point to another by using the Euclidean formula and then we multiplied it by the velocity of the car, for this project we used a velocity of 1 for simplicity. Continuing, we also had to consider that once the worker was in the station it would take a certain amount of time to deliver or pick up the package, therefore we added this process to the total time. Additionally, while doing these routes, there were pick-up stations or delivery stations that didn't open after a certain amount of time. Meaning, that if the car arrives at this station earlier than the time it opens, the car will have to wait until it opens, adding this pause to the total time.

After considering the time, we continue with capacity. This variable acted as a constraint on how much the car could carry, meaning for this scenario the car can't carry more than its limit. Therefore, if the car exceeds the capacity, it will return to very high fitness so that the algorithm wouldn't consider this route as an option. In simpler terms, when the car arrives at a station it will add or decrease the number of packages it carries, depending on if it is a pickup or delivery station, and if this exceeds its maximum capacity it returns a very high number.

For our next step, we had to contemplate the vehicle's battery. This is because every time a car heads from one station to another, an amount of battery is consumed. This is calculated by the distance from one point to another times the battery consumption, then the result is subtracted from the vehicle's battery. Additionally, a car should not consume more than its battery capacity, as the car can't move after it is out of battery. Therefore, in the fitness function whenever a car is heading to a new station, we calculate its energy consumption from its actual location to the next point and if it ends up being less than zero it means that it runs out of battery. Thus, before it goes to that station, the car needs to head up to the nearest charging station. We calculate the distance from where the car is to every recharging station and add it to the distance from the recharging station to the station it needs to go. The recharging station that gives the least distance is where the car will head up. This way, the car will always have a battery and will try to always use the shortest amount of time for recharging. Furthermore, it is important to note that the time it takes to visit the charging station and go to the next point is added to the total amount of time.

Continuing, an important variable and constraint in this project is the due date, the time the delivery needs to give the package to the customer. In the beginning, we thought of adding an if statement that if the car arrives at a delivery station after the due date the fitness will be a very high number, making the algorithm not choose this route. However, after analyzing the dataset and several attempts at running the code, we concluded that this was not the best approach, because it tends to just create only one route for one car. We then decided to create a variable called delay time. This variable

is calculated when the car arrives at a station after its due date and is the difference between the current time and the due date, giving how much time the car arrived late. As delay is an important factor we decided to add weight.

Finally, we want to consider not only reducing the amount of time each car takes but also reducing the number of cars the companies use in each scenario. We decided to create a sum of the time it takes on each route and then multiply it by the number of cars being used. This is to make the algorithm consider the number of cars as a weight and decide the best routes while at the same time reducing the number of cars. If this had not been implemented, the algorithm would have used the maximum number of cars the company has, as each car would then have fewer stations to go.

In conclusion, the following formula is the one we used for fitness, if you want to see it in more detail there is a more detailed formula in the appendix [Formula 2, Formula 3]:

$$Fitness = (\sum((Time \times 0.3) \ + \ (Delay \times 0.7)) \times Cars) \ + \ Capacity$$

Formula 1. Fitness Function

Here time is the total amount of time being used, the delay is how much time the car arrives late, and capacity is either a 0 if it didn't reach its maximum capacity or 10.0000.000 if it reaches its capacity, and cars are the total number of cars in this scenario. Lastly, we gave a weight for time and delay, as we are also trying to prevent any delays, we decided to give a higher weight on delay and a lower weight on time. However, if there is no delay then the delay is 0, making a better fitness for the algorithm to choose.

To illustrate the effectiveness of our fitness function, Figure 0 shows an example of the resulting routes. With this fitness function, we can guarantee feasible solutions that visit each location, use the minimum number of cars needed, respect the capacity restriction, and, if necessary, include trips to charging stations to maintain battery levels. This plot demonstrates the practical application of our approach, ensuring that all constraints are met efficiently.

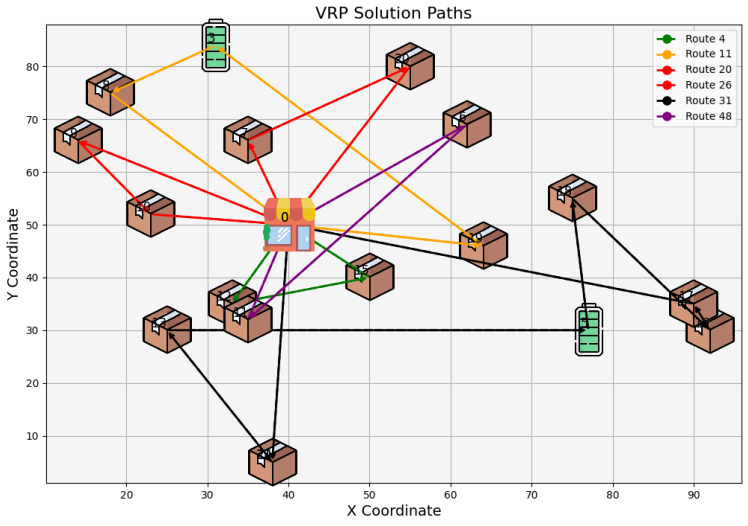


Fig 0. Example of the routes

3. Results

Our implementation results were evaluated across various configurations. These configurations, each with unique combinations of crossover, mutation, and selection methods, were tested. We conducted 100 generations and 20 iterations for each configuration, calculating the median of the best fitness values over multiple runs. To provide a comprehensive analysis, we compared our results on two different instances: one with 106 pickup or delivery locations and another with 16 locations. The detailed results for each instance are presented in Figure 1 to Figure 4, respectively.

To determine our best configuration, we considered the one that minimizes our fitness function, which means the one that, at the final generation, minimizes the amount of time and the number of cars and respects the condition of capacity. The configurations that worked best together are: tournament selection, single-point random crossover, and swap mutation. The intuition behind tournament selection provides a good balance by ensuring high-quality solutions are more likely to be chosen while still allowing for some diversity in the population. Single-point random crossover effectively combines genetic material from parents while preserving substructures, ensuring feasible offspring, which is crucial for our problem. Swap mutation introduces minor, effective variations that help maintain diversity within the population. Together, these methods lead to efficient convergence and good performance across our two instances.

However, it is also important to analyze the variance of our fitness function to assess the robustness of the configurations. So, we analyzed the variance in the last generation across the 20 iterations. As shown in Figure 7 and Figure 8, our best configuration (tournament selection, cross-validation single point random, and swap mutation) exhibits low variance. This indicates that it is a robust configuration and provides better stability and consistency in the results.

We also analyzed the impact of inclusion or exclusion of elitism. Implementing elitism in our algorithm should accelerate the convergence. The intuition behind this is to preserve high-quality solutions. Having this in mind, as we can see in Figure 5 and Figure 6, when we use elitism, the fitness variance is controlled, and it helps all configurations to converge. So, for our solution, it is highly recommended that we use it since it helps maintain stability in the fitness function.

4. Conclusions

In this project, we successfully solve the vehicle routing problem, by applying GA, even with the added complexity of electric vehicle constraints. Despite the challenges, the algorithm effectively minimized delivery distances and balanced multiple objectives. By experimenting with various configurations of selection, crossover, and mutation methods, we identified that a combination of tournament selection, single-point random crossover, and swap mutation, enhanced with elitism, provided the most effective and consistent results.

Our results showed that this configuration minimized the total travel distance and the number of vehicles used, while also respecting vehicle capacities. The incorporation of elitism played a crucial role in convergence and maintaining solution quality by preserving the best solutions across generations. Furthermore, GA has the flexibility to account for several constraints, like in this case vehicle capacities, delivery time windows, and battery level.

5. References

Rahmat, R. W., & Zaharuddin, W. M. (2013). Solving the Vehicle Routing Problem using Genetic Algorithm.

Toth, P., & Vigo, D. (2002). An overview of vehicle routing problems. European Journal of Operational Research, 144(3), 465-474.

Keskin, M., & Çatay, B. (2018). The electric vehicle routing problem with time windows and recharging stations. Transportation Research Part C: Emerging Technologies, 87, 113-137.

Michael Schneider, Andreas Stenger, Dominik Goeke (2012). The Electric Vehicle Routing Problem with Time Windows and Recharging Stations.

Goeke 2018 dataset: http://www.vrp-rep.org/datasets/item/2019-0001.html

6. Annex

Dataset:

Each row of the dataset contains the following columns:

- **StringId:** Unique identifier of the location;
- **Type:** Indicates the function of the location, where:
  - **D:** depot;
  - **F:** recharging station;
  - **CP:** customer pickup location;
  - **CD:** customer delivery location.
- **X, Y:** Location coordinates (distances are assumed to be Euclidean);
- **Demand:** Specifies the quantity of freight capacity required (positive at pickup, negative at delivery);
- **ReadyTime and DueDate:** beginning and end of the time window (waiting is allowed);
- **ServiceTime:** denotes the entire time spent at pickup/delivery for loading/unloading operations;
- **PartnerId**: Relevant for transportation requests and provides the StringId of the partner of each pickup and delivery location.

Besides the locations, the dataset also contains information about the vehicle, such as velocity, freight capacity, and battery capacity.

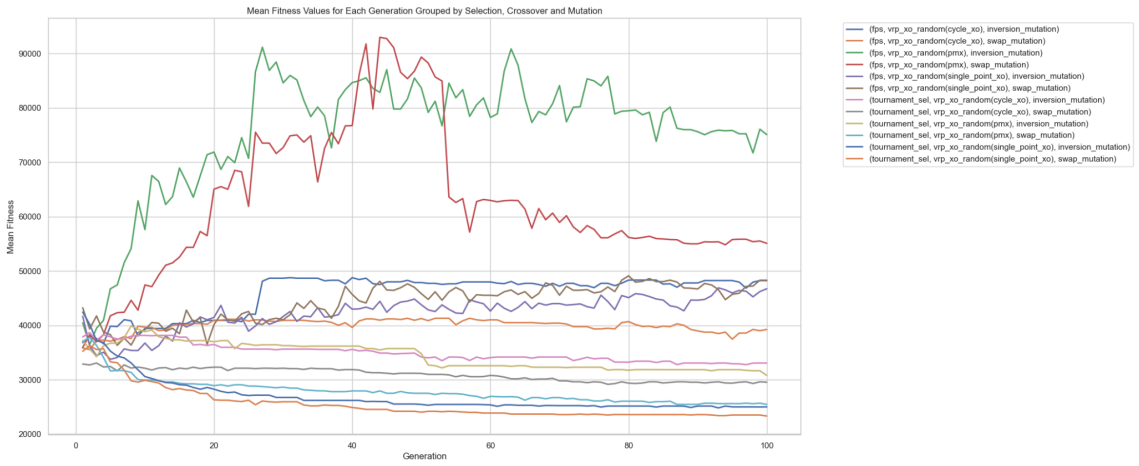


Fig 1 Instance lc101 without elitism and random

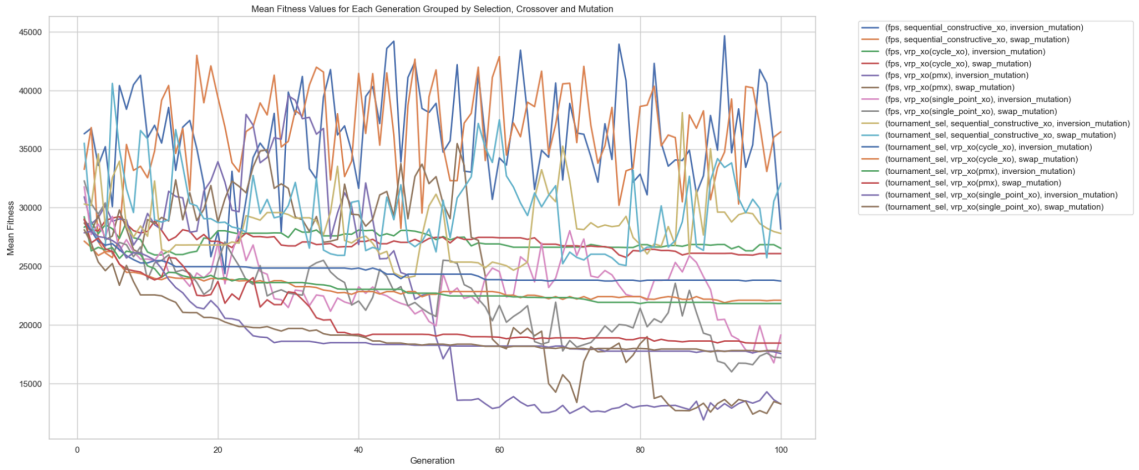


Fig 2 Instance lc101 without elitism and not random

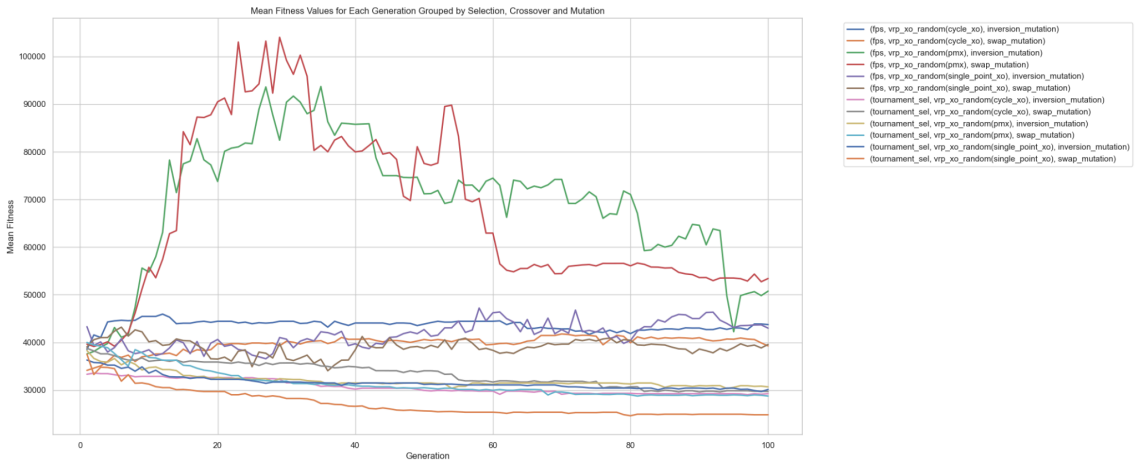


Fig 3 Instance c202C16 without elitism and random

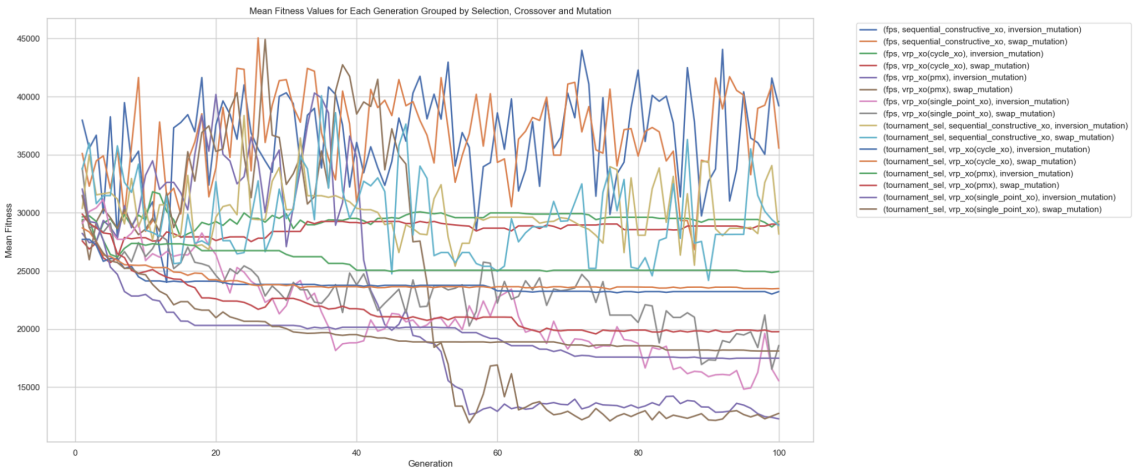


Fig 4 Instance c202C16 without elitism and not random

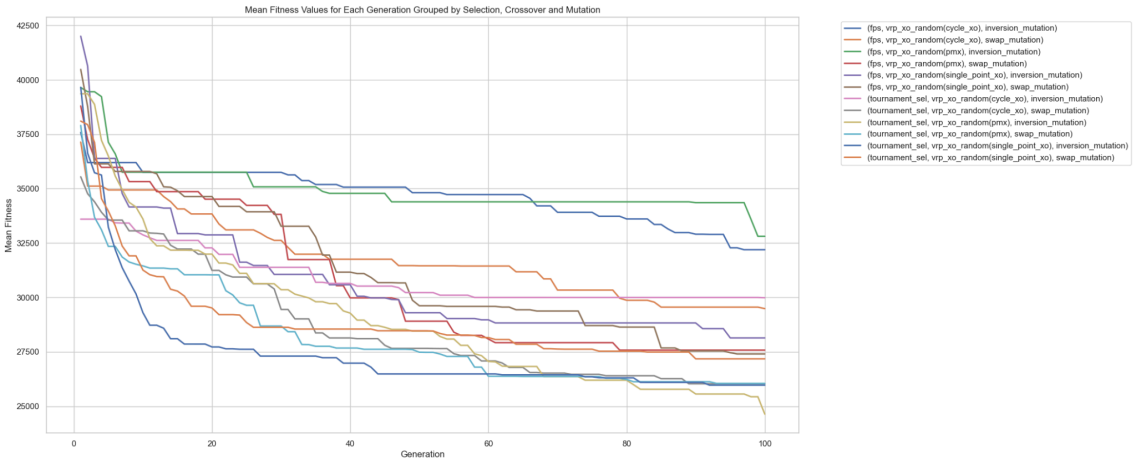


Fig 5 Instance lc101 with elitism and random

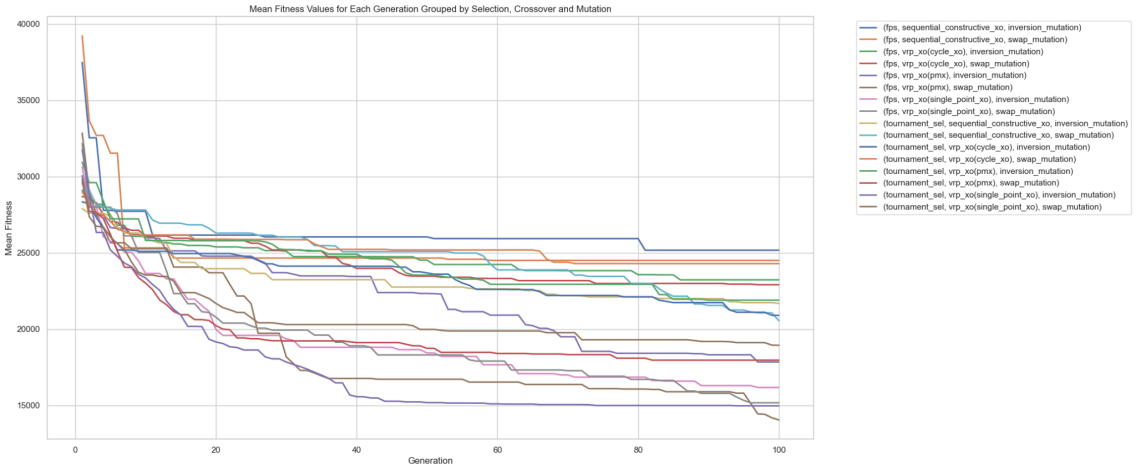


Fig 6 Instance lc101 with elitism and not random

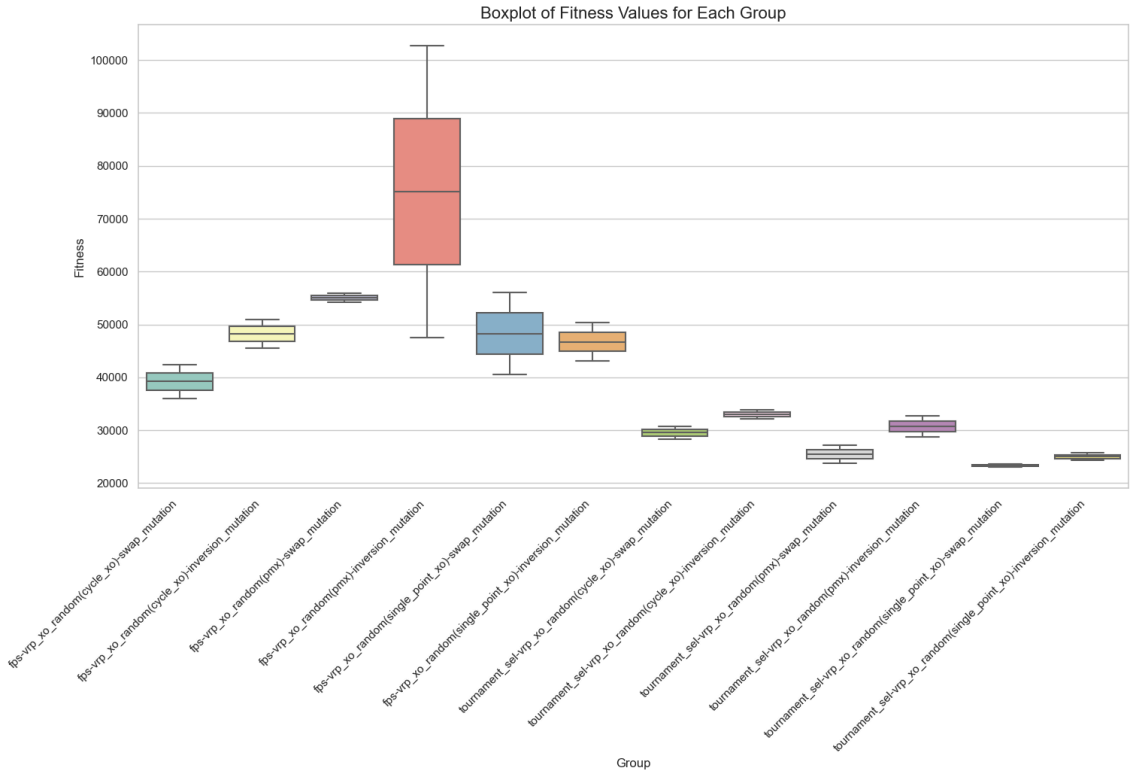


Fig 7 Instance lc101 last generation across the 20 iterations

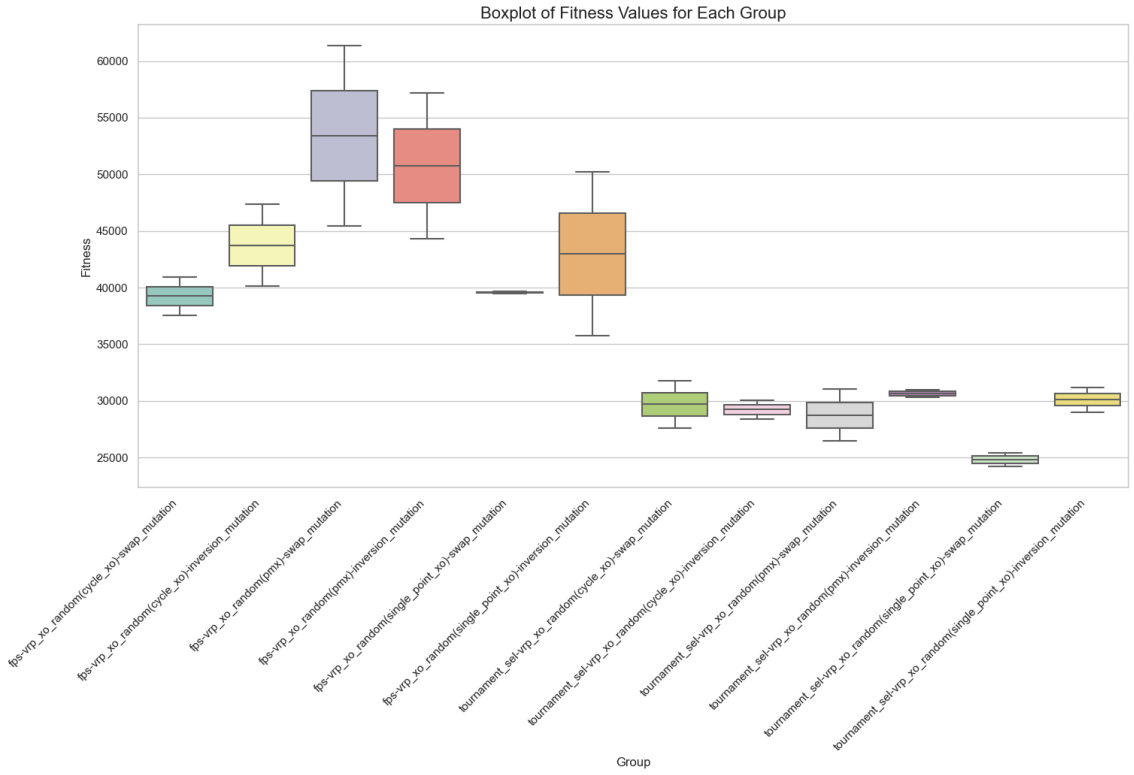


Fig 8 Instance c202C16 last generation across the 20 iterations

$$Euclidean\ Distance(d) = \sqrt{(X_2 - X_1)\hat{}2 + (Y_2 - Y_1)\hat{}2}$$

Formula 2. Euclidean Distance

$$Time(point\ A\ to\ point\ B) = EucledianDistance(A, B) \times (Velocity\ of\ Car)$$
$$Time = Time(point\ A\ to\ point\ B) + (Time\ to\ pick\ or\ deliver\ package) + (Waiting\ Time\ to\ Open) + (Battery\ Charge\ Time)$$
$$Battery\ Charge\ Time = (Time\ form\ Point\ A\ Charge\ Station) + (Time\ form\ Point\ B\ Charge\ Station)$$
$$Delay = Current\ Time - Duedate$$
$$Waiting\ Time\ Open = Current\ Time\ Becomes\ to\ Time\ it\ Opens$$

Formula 3. Calculation Details for Fitness Function