A metaheuristic approach to the Capacitated Electric Vehicle Routing Problem with Time Windows (E-CVRP-TW)

Juan Betancourt [a], Daniel Giraldo [b]

[a] Universidad de los Andes, Bogotá, Colombia, jm.betancourt@uniandes.edu.co

[b] Universidad de los Andes, Bogotá, Colombia, ds.giraldoh@uniandes.edu.co

Abstract

We propose a specialized metaheuristic based on hybrid genetic algorithm to resolve Capacitated Electric Vehicle Routing Problem with Time Windows and Recharging Stations (E-CVRP-TW). This problem incorporates servicing a set of clients with a fleet of electrical capacitated vehicles with the possibility of recharging at any of the available stations using an appropriate recharging scheme. Benchmark instances of related problems were tested to demonstrate the high performance of the proposed metaheuristic as well as the positive effect of the hybridization of a Genetic Algorithm with local search (and solution repair) operators.

Keywords`

Capacitated Electric Vehicle Routing Problem, metaheuristics, approximate algorithms, hybrid genetic algorithm

1. Introduction

Due to the popularity of electric vehicles and policies to decrease the carbon footprint, organizations have opted to change their fleet from internal combustion vehicles based on fossil fuels to fleets of electric vehicles. Despite the great catch they have, these vehicles contemplate several operational challenges that conventional fleets do not have. For example, the distances these vehicles can travel are shorter than combustion vehicles and the charging network for these vehicles is not as developed as internal combustion vehicles. Also, the time required to charge these vehicles can range from 30 minutes to several hours. (Pelletier, Jabali, & Laporte, 2016).

The usual distribution tasks performed by a logistics company can be represented as vehicle routing problems (VRP) in which they seek to minimize total transportation costs by visiting a set of customers from a depot and returning to the same depot at the end. Over the years multiple variants have been generated for these VRPs, the most usual ones are that the cargo vehicles have a limited capacity called CVRP (Cordeau, Laporte, Savelsbergh, & Vigo, 2007). Additionally, when the customers have a time interval in which they can receive the goods or service is called VRPTW. This restriction changes the visiting dynamics, now the vehicle must be routed based on availability time windows. If a vehicle reaches a costumer before the time windows starts, it needs to wait in order to service the costumer when the window starts. The objective of the VRPTW is to service all customers without violating vehicle capacity and time window constraints with a minimum number of vehicles and, for the same number of routes with the minimum travel distance, followed by the minimum schedule time and the minimum waiting time (Li & Lim, 2003).

Nowadays electric vehicles have higher performance and can travel longer distances on a full charge, to make these vehicles sustainable, an energy supply network must be in place so that they can be efficiently harnessed, however, contemplating these charging times can be detrimental to the delivery of orders to customers. In this paper we present a metaheuristic to solve the electric vehicle routing problem with time windows and charging stations (E-CVRPTW), which incorporates the possibility of recharging the vehicle battery at a charging station for any instant of the route without the need to return to the depot, considering capacity constraints, service time and customer time windows.

This article presents on section 2 the problem definition, and a literature-coherent mathematical formulation is presented. On section 3, the genetic algorithm with the methodology and operators are defined in a pseudo-code and explained to the detail. On section 4, the instances used on the experiments are referenced and all the conditions of the experimentation are defined. On section 5, the results are shown, and a corresponding analysis is performed. Finally, on section 6 conclusions are made over the work and results and section 7 present areas of future work.

***All coding resources, instances, results and graphs for all tested instances are available on GitHub through https://github.com/juanbeta98/CG-VRP-TW**

2. Problem definition and mathematical formulation

The critical factor of electric vehicles is energy, the energy consumption of these vehicles and any other in general depends on the power required to move depending on the weight of the load, to accelerate the vehicle, when there are sloping roads. The E-VRPTW model proposed by (Schneider, Stenger, & Goeke, 2014) contains some considerations such as that the deliveries are made on a flat terrain and without inclines, it does not consider the increase in battery consumption if the vehicle is carrying a load or not. The speed of the vehicle is constant between vertices. All these factors are known to be important in measuring battery consumption and determining whether the vehicle should visit recharging stations, how much it should recharge and how long it will take to do so.

2.1 Problem definition

It is defined in a complete directed network $G = (V_{N+1}, A)$ with a set of arches $A = \{(i,j)|i,j \in V'_{0,N+1}, i \neq j\}$. For each arc there is an associated distance $d_{i,j}$ and a travel time $t_{i,j}$. Traveling through an arc i,j consumes a portion of the battery's energy $h * d_{i,j}$, being h an energy consumption rate constant. The vehicle fleet is homogeneous and has a charging capacity C, this vehicle flora always starts from the depot. Each vertex in the network except for the charging stations has a positive demand of q_i, also has a time window $[e_i, l_i]$ in which it can be serviced, all these vertex $i \in V_{0,N+1}$ have a service time s_i which cannot start before e_i, so if the truck arrives before the time window starts it must wait, and it is not possible to be serviced after the end of the time window l_i, however, the service time s_i can end after the time window. At charge stations, the vehicle will recharge until complete charge Q based on his actual charge, with a rate charge g that recharge the vehicle linearly.

2.2 Mathematical formulation

$0, N+1$	Depot instances
F'	Set of visits to recharging stations, dummy vertices of set of recharging stations F
F'_0	Set of recharging visits including depot instance 0
V	Set of customers V = {1, … , N}
V_0	Set of customers including depot instance 0
V'	Set of customer vertices including visits to recharging stations, $V' = V \cup F'$
V'_0	Set of customers and recharging visits including depot instance 0: $V'_0 = V' \cup \{0\}$
V'_{N+1}	Set of customers and recharging visits including depot instance $N+1$: $V'_{N+1} = V' \cup \{N+1\}$
$V'_{0,N+1}$	Set of customers and recharging visits including depot instances 0 and N+1: $V'_{0,N+1} = V' \cup \{0\} \cup \{N+1\}$
$d_{i,j}$	Distance between vertices i and j
$t_{i,j}$	Travel time between vertices i and j
C	Vehicle capacity
g	Recharging rate
h	Charge consumption rate
Q	Battery capacity
q_i	Demand of vertex $i, 0\ if\ i \notin V$
e_i	Earliest start of service at vertex i
l_i	Latest start of service at vertex i
S_i	Service time at vertex i $(S_0, S_{N+1} = 0)$
τ_i	Decision variable specifying the time of arrival at vertex i
U_i	Decision variable specifying the remaining cargo on arrival at vertex i
y_i	Decision variable specifying the remaining battery capacity on arrival at vertex i

x_{ij} Binary decision variable indicating if arc (i, j) is traveled

$$\sum_{i \in V_0', j \in V_{N+1}', i \neq j} d_{ij} x_{ij} \tag{1}$$

$$\sum_{j \in V_{N+1}', i \neq j} x_{ij} = 1 \ \forall i \in V \tag{2}$$

$$\sum_{j \in V_{N+1}', i \neq j} x_{ij} \leq 1 \ \forall i \in F' \tag{3}$$

$$\sum_{j \in V_{N+1}', i \neq j} x_{ij} - \sum_{i \in V_0', i \neq j} x_{ij} = 0 \ \forall j \in V' \tag{4}$$

$$\tau_i + \left(t_{ij} + s_{ij}\right) x_{ij} - l_0\left(1 - x_{ij}\right) \leq \tau_j \quad \forall \ i \in V_0, \forall j \in V_{N+1}', i \neq j \tag{5}$$

$$\tau_i + t_{ij} x_{ij} - g(Q - y_i) - (l_0 + gQ)(1 - x_{ij}) \leq \tau_j \ \forall \ i \in F', \forall j \in V_{N+1}', i \neq j \tag{6}$$

$$e_j \leq \tau_j \leq l_j \ \forall j \in V_{0,N+1}' \tag{7}$$

$$0 \leq u_j \leq u_i - q_i x_{ij} + C\left(1 - x_{ij}\right) \forall i \in V_0', \forall j \in V_{N+1}', i \neq j \tag{8}$$

$$0 \leq u_0 \leq C \tag{9}$$

$$0 \leq y_i \leq y_i - \left(h * d_{ij}\right) x_{ij} + Q\left(1 - x_{ij}\right), \forall j \in V_{N+1}', i \in V, i \neq j \tag{10}$$

$$0 \leq y_i \leq Q - \left(h * d_{ij}\right) x_{ij}, \forall j \in V_{N+1}', i \in F_0', i \neq j \tag{11}$$

$$x_{ij} \in \{0,1\} \ \forall i \in V_0', \forall j \in V_{N+1}', i \neq j \tag{12}$$

Equation (1) is the objective function which seeks to minimize the distances traveled. Constraint (2) makes it must visit customers; constraint (3) manages the visit of vehicles to charging stations. Constraint (4) guarantees flow conservation in the network, where a vehicle cannot stay at a customer vertex or charging station, the number of incoming arcs must be equal to the number of outgoing arcs. Constraint (5) guarantees the feasibility of time windows for customers visited from the depot and constraint (6) guarantees the feasibility of serving customers from the refueling stations. Constraint (7) guarantees that I can serve the customer only within the time window he/she has. Constraints (8) and (9) guarantee the fulfillment of the demand of all customers in the network considering truck loads and non-negativity. Constraints (10) and (11) make sure that the electric vehicle charge never reaches zero; it is necessary to clarify that every time a vehicle arrives to recharge at a station, it will charge the battery to the maximum. Constraint (12) presents the natures of the variables.

3 Solution Approach

As a solution for the Capacitated Electrical Vehicle Routing Problem with Time Windows, we propose a genetic algorithm scheme with several crossover and mutation operators for the routes. The algorithm combines elitism,

random sampling with a fitness function, and tournament to exploits three key ideas: (1) the use of a population of solutions to guide search, (2) the use of crossover operators that recombine two or more solutions to generate new and potentially better solutions, and (3) the active management of diversity to sustain exploration. New ideas that are also introduced in this chapter include (1) the use of deterministic recombination operators that can tunnel between local optima, and (2) the use of deterministic constant time move operators (Gendreau & Potvin, 2018). The E-CVRP-TW has constraints that makes the problem challenging and defiant; even evaluating a single route can be a hard task from computational burden, due to the need of verifying feasibility of energy, visit the charging stations, arrival times, and capacity of EV.

3.1 Overall description

Figure 1. Represents the general scheme of the genetic algorithm implemented in this work. First, an initial population of solutions is generated through a constructive heuristic. Each one of the solutions comprehends the entire routing scheme for the instance and has an associated objective value (i.e., the total distance of the routes). Through this objective, the individuals are ranked and undergo random weighted sampling to achieve an intermediate population. This intermediate population is somewhat biased with an elite list, the best individuals of the population. From the intermediate population, again through random sampling and a tournament, two parents are selected. In this work, from two parent solution only one offspring solution is obtained. Therefore, the number of couple of parents must be the same as the number of individuals in the population. From a couple of parents, an offspring can go through cross-over or/and mutation operators which ensemble the new individual of the next generation. Due to the large number of constraints and the feasibility's sensibility to change in solutions, most of the operators will produce unfeasible solutions. Therefore, after the perturbations to the offspring, a local search is done with a reconstructive/repair operator. With feasibility guaranteed on each of the new individuals, the new population will undergo the process again.

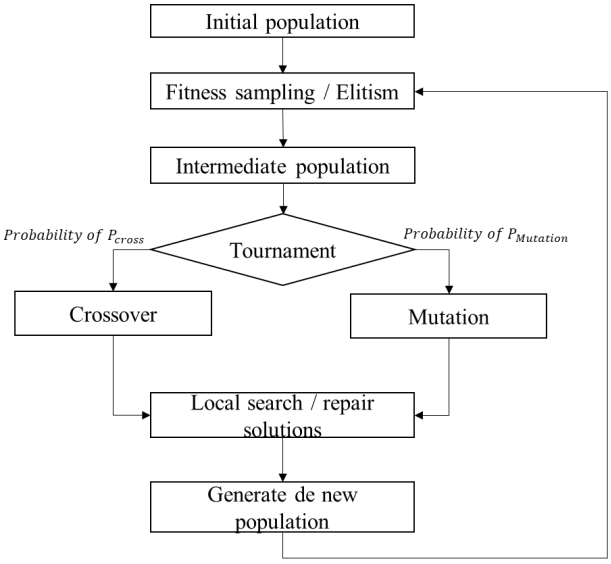


Figure 1. Overall view of the methodology

3.2 Initial population

Before the genetic algorithm can take place, an initial pool of feasible solutions (population) is required. This population must assure diversity in solutions to promote the exploration of the solution space. Furthermore, we require feasible routes, based in three different criteria: (1) the EV can arrive at costumer c if it can be attended in his time window, (2) the EV has enough energy, then can reach at least a charging station after visits a costumer, (3) the EV has enough capacity load to supply the demand at costumer c. For this, we designed a Restricted-Candidate-List (RCL)-based heuristic founded in the approach proposed by Cordeau, Laporte and Mercier (2001). The heuristic is randomized to achieve a high level of diversity. A pseudo-code for this heuristic is presented in Algorithm 1.

Algorithm 1 RCL based constructive

Input: α, v, r, g, $dist$, $closest$, RT, DD, ST, mode, slack
Output: t, q, k, route

$t \leftarrow 0$
$q \leftarrow Q$
$k \leftarrow 0$
$node \leftarrow Depot$
$route \leftarrow \{node\}$
$stable \leftarrow$ **true**

while $stable$ **do**
  $target, energy\ feasible \leftarrow generate\ candidate\ from\ RCL(node, t, q, k, v, r, d, v, r, dist, \alpha, \text{mode, slack})$
  if $target \neq$ **false then**
    $route, t, q, k \leftarrow route\ costumers(node, target, route, t, q, k, v, r, d, dist)$
    $node \leftarrow target$
  else
    if $energy\ feasible$ **then**
      closest station $\leftarrow closest(node)$
      $t \leftarrow t + dist(node, \text{closest station})/v$
      $t \leftarrow t + (Q - q) * g$

      $q \leftarrow Q$
      $node \leftarrow$ closest station

      $target, energy\ feasible \leftarrow generate\ candidate\ from\ RCL(node, t, q, k, v, r, d, v, r, dist, \alpha, \text{mode, slack})$
      if $target \neq$ **false then**
        $route, t, q, k \leftarrow route\ costumers(node, target, route, t, q, k, v, r, d, dist)$
        $node \leftarrow target$
      else
        $route, t, q, k \leftarrow route\ to\ depot(t, q, k, node)$
      end if
    else
      $route, t, q, k \leftarrow route\ to\ depot(t, q, k, node)$
    end if
  end if
end while
return t, q, k, route

Algorithm 1. RCL-based constructive heuristic.

The constructive heuristic requires the following parameters: alpha, the speed, the energy consumption and recharge rates, the distances, the closest station to each costumer, the ready, due, and service times of each costumer, the criterion to restrict the list of candidates, and finally, a slack parameter for routing termination. The heuristic returns

the total time, final charge, final load, and route. Lines 1-5 initialize time, charge, load, and route (starts at depot). While the route has inserting options (Line 6) The $generate\ candidate\ from\ RCL$ function returns a target node to append to the route or *False* if there is no feasible candidate. If there is a feasible candidate (Lines 9-11), it is appended to the route and the metrics are updated. If there is not (Line 12), the variable $energy\_feasible$ stores whether there is at least one candidate feasible by time and load but not by charge (not enough to reach). If there is (Line 13), the vehicle is routed to the nearest station and charged fully (Lines 14-18). Then, a new candidate is selected with $generate\ candidate\ from\ RCL$. If there is, it will be visited next (Lines 20-22). If there is not (Lines 23-24), or there is no energy feasible candidate (Lines 26-27), the vehicle is routed to the depot.

Algorithm 2 generate candidate from RCL

Input: $node, t, q, k, v, r, d, dist, \alpha$, mode, slack
Output: Updated time, charge and load, route

1: $feasible\ candidates \leftarrow \{\}$
2: $crit_{max} = -M$
3: $crit_{min} = M$
4: $energy\ feasible \leftarrow$ **false**

5: **for** $target\ in\ pending\ costumers$ **do**
6: $feasible, energy\ feasible \leftarrow evaluate\ candidate(candidate)$
7: **if** $feasible$ **then**
8: $feasible\ candidates \leftarrow feasible\ candidates \cup \{target\}$

9: $crit \leftarrow$ creterion defined by parameter mode($target$)
10: $crit_{max} \leftarrow$ máx($crit, crit_{max}$)
11: $crit_{min} \leftarrow$ mín($crit, crit_{min}$)
12: **end if**
13: **end for**
14: **if** $feasible\ candidates = \{\}$ **then**
15: **return false**, $energy\ feasible$
16: **else**
17: $upper\ bound \leftarrow crit_{min} + \alpha(crit_{max} - crit_{min})$
18: $RCL \leftarrow$ candidates with criterion $<$ $upper\ bound$
19: $target \leftarrow$ selected random from RCL
20: **return** $target, energy\ feasible$
21: **end if**

Algorithm 2. Constructing the RCL and selecting the candidate.

The pseudo-code on Algorithm 2. details how the Restricted List is constructed, and the candidate is selected. This algorithm the following inputs: the current node, the current time, charge and load, other instance parameters, the accumulated distance, the distances, alpha, the criterion to restrict the list and the slack. It returns the updated time, charge, load, and route. Lines 1-4 initialize the list of candidates, the minimum and maximum values of the criterion and the Boolean existence of an energy feasible candidate. In this work, two different criteria are considered: distance and time window finalization. For all unvisited costumers (Line 5), the candidate's feasibility is evaluated (Line 6). The detailed criteria are:

1) Time window: The accumulated time plus the travel time to the candidate is less than the due date of the service for that costumer.
2) Energy: The vehicle has enough energy to visit the candidate and attend to recharge to the nearest station.

3) Capacity: The vehicle has enough load capacity to service the candidate

If the candidate satisfies all three conditions, it will be appended to the restricted list (Line 8). Then, the designed criterion of the candidate is stored, and the maximum and minimum criteria values are updated (Lines 9-11). If there are no feasible candidate (satisfies all three conditions), then the algorithm returns *false* and the $energy\_feasible$ variable (Lines 14-15). If there is at least one feasible candidate (Line 16), an upper bound is computed using the maximum and minimum values found, as shown in Line 17. Then, the candidate list is assembled with those candidates whose criterion value is less or equal to the upper bound computed (Line 18) and a candidate is selected randomly from the list (Lines 19-20).

As for the other two functions displayed on Algorithm 1, there will be no pseudo-code shared but the main logic of the algorithms is presented below:

- *route costumers:* As it is used on fully feasible transitions, this function updates the time with the travel and service times, updates the charge with the loss of energy on the transportation, updates the load with the costumer's demand and updates the route by adding the costumer to the route.
- *route to depot:* If the vehicle can transport from the current node to the depot with the current charge, then it's routed directly to the depot and the time, charge and route are updated accordingly. If there isn't enough energy, the closest station (best station) to both the current node and the depot is computed. If the vehicle has enough energy to reach this station, it's added to the route and the time and charges are updated accordingly. If the vehicle cannot reach the best station, the vehicle is routed through the closest station to the current node and to the depot.

3.2 Intermediate population

After an initial feasible population is designed, the genetic process begins. The first step on every iteration is to generate an intermediate population from which the parents are going to be selected. This will somewhat assure that the recombination and mutation processes will be applied on the best individuals. This intermediate population is configured with the same size and individuals as the initial population. However, the order and quantity of everyone on the intermediate population varies. The placement of individuals into the intermediate population has been randomized, adjacent individuals can be recombined (Gendreau & Potvin, 2018).

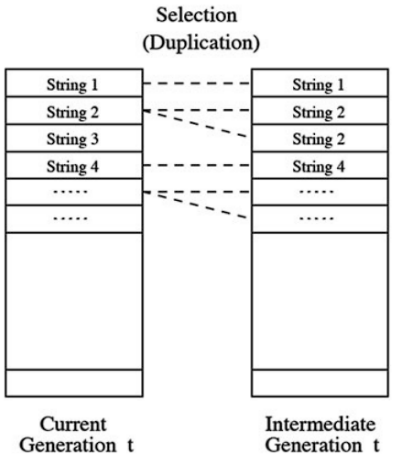


Figure 2. Constructing the RCL and selecting the candidate

3.2.1 Elitism

As the genetic algorithm progresses, it would be desirable to build a list with the best-known individuals and somehow prevail them to seek local searches y promisor neighborhoods. This is exactly the concept attacked by an elite class. On each iteration, the best 5% individuals are prevailed to the intermediate population.

3.2.2 Fitness sampling

The remaining 95% individuals of the intermediate population are randomly sampled out of the initial population (including the elite individuals). To do this, a fitness function is computed for each individual as shown in Equation 1.

$$f_i = \frac{\sum_{j \in Population} d_j}{d_i} \qquad \forall \qquad i \in Population$$

Equation 1. Computation of the fitness function

Were, for every individual i of the initial population, the fitness function will be larger for individuals with smaller distances and smaller for individuals with larger distances. With this fitness function, the probability of choosing each individual to the intermediate population on each sample is computed as shown in Equation 2.

$$p_i = \frac{f_i}{\sum_{j \in Population} f_j} \qquad \forall \qquad i \in Population$$

Equation 2. Computation of the probability based on fitness

Hence, the probability for individuals with smaller distances (objective) will have a larger probability of being chosen for the intermediate generation.

3.3 Tournament

From the intermediate population, the pairs of individuals to recombine and mutate are chosen through a tournament. To generate each pair of parents, two individuals are chosen at random and the best one, ergo tournament, is selected

to be one of the parents. For the second parent, the same process is applied. As mentioned in Section 3.1, one pair of parents will produce one offspring. Hence, there will be as many pairs of parents as individuals on the population.

3.4 Crossover

To generate an individual from a pair of parents, the first phase are the *crossover* operators. This operator chose one of the parents at random and operates this solution. Any operations realized over a solution will for surely modify the existing charging stations and time window feasibilities. To avoid spending too much effort on assuring feasible *crossover* and *mutation* operators, the feasibility requirement is left aside until the repair phase of the scheme. For any of the different operators presented in Sections 3.3.1 – 3.3.5 we modify the codification of the problem to simplify the operations. Fist, all the depots and the stations are removed from the routes. Then, all the routes are merged into one single vector. As the costumers are visited once, this vector will have as many positions as costumers and will be a permutation of the costumers. Figure 2 exemplifies this transformation for a two-route individual.

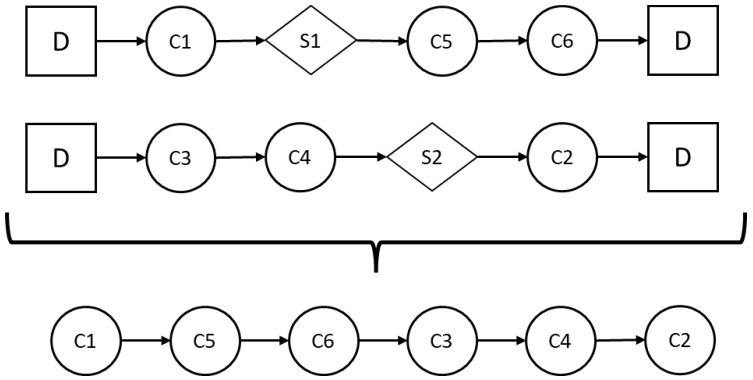


Figure 2. Change of codification

3.4.1 Simple crossover

The first operator a swap between two positions of the modified vector. For this operation, two different positions of the vector are chosen at random. The element of the first position is replaced by the item on the second position and vise-versa. Figure x shows this operation on the example modified vector with positions one and two being 2 and 5 (on bold), respectively. The costumer ‘C5’ is on position 2 in the parent, and it’s placed on position 5 on the offspring. Similarly, costumer ‘C4’ is on position 5 on the parent and it’s placed on position 2 on the offspring.

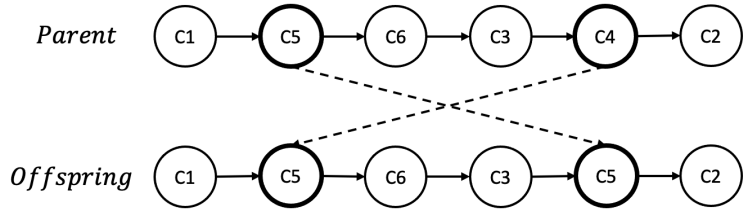


Figure 3. Simple crossover operator

.

3.4.2 2-opt

This operator is somewhat more complex than its predecessor. Again, two positions are chosen at random from the vector. However, these positions must be at least 2 positions apart. The new vector will remain with the first section (from the start to the first position) and the final section (from the second position to the end). As for the middle section, the costumers will be reversed in order. Figure x shows the operator on the vector example. For the example, positions will be 1 and 6 (first and last). Therefore, these two positions will remain the same and the nodes in between ('C5', 'C6', 'C3' and 'C4') will be reversed in order.

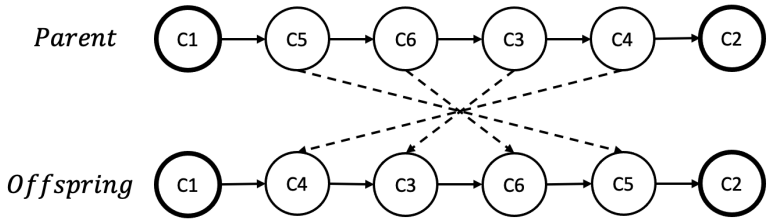


Figure 4. 2-Opt operator

3.4.3 Smart crossover

The smart crossover follows the same philosophy that the simple crossover. However, it has an additional level of complexity. One position of the modified vector is chosen at random. Then, a *compatibility index* is computed for each one of the other costumers of the vector. This index takes two main components into account: difference of overlap of Time-Window and distance of the change. Equations 2, 3 and 4 showcase the computation of the two components and the index for a first position $i \in Costumers$ and all positions $j \in Costumers | i \neq j$.

$$overlap = \left|DueDate_i - DueDate_j\right| + \left|ReadyTime_i - ReadyTime_j\right|$$

$$distance = dist(j-1, i) + dist(i, j+1) + dist(i-1, j) + dist(j, i+1)$$

$$index_j = overlap + distance$$

Finally, the position j with a lower index is chosen and the two costumers, at positions i and j, are swapped.

3.4.4 Hybrid

The Hybrid operator chooses one of the 3 defined crossover operators at random on every iteration.

3.5 Mutations

3.5.1 Simple insertion

As a mutation, we propose a simple insertion operation. For this, operator we based the nearest costumer between two nodes and verify if that new distance is shorter than the currently distance. (Vidal, Crainic, Gendreau, Lahrichi, & Rei, 2012)

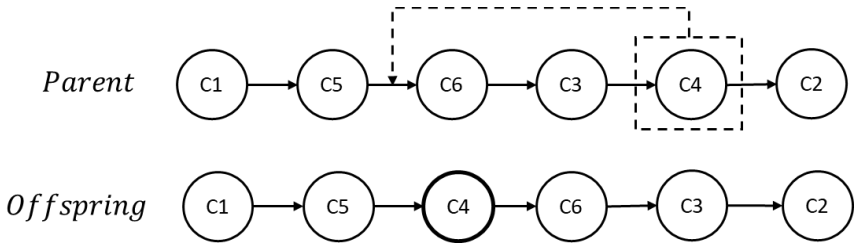


Figure 5. Simple insertion operator

3.6 Reconstruction / Repair

As mentioned vastly on previous sections, all operations realized over a vector will most probably affect the feasibility of the solution in one of the dimensions. Therefore, a consistent and reliable repair operator is needed. This operator will take as input one of the modified vectors (a permutation of the costumers) and reconstruct the route/s whenever feasibility is compromised in one of the transitions between edges. This means, the operator must be able to route to a station, route to a different costumer or route to the depot to avoid unfeasible routes. Algorithm 3 has the pseudo-code for this repair operator.

Algorithm 3 reconstruct modified solution vector

Input: *modified solution*
Output: routes, t, q, k

```
1: parent ← {}
2: distance ← 0
3: time ← 0
4: pending ← {}
5: i ← 0
6: while i < |modified solution| − 1 do
7:    modified solution ← modified solution ∪ pending
8:    pending ← {}
9:    route ← {'D'}
10:   t, d, q, k ← 0, 0, Q, 0
11:   node ← modified solution(i)

12:   route, t, d, q, k ← route costumers('D', node, t, q, k, v, r, d, dist)
13:   route done ← false
14:   while route done = false do
15:      target ← modified solution(i + 1)
16:      if not capacity (load) feasible then
17:         t, d, q, finish route to depot(node, t, d, q
18:         route ← route ∪ finish route
19:         i+ = 1
20:         route done = true
21:      else if not total time feasible then
22:         i− = 1
23:         node ← modified solution(i)
24:         t, d, q, finish route to depot(node, t, d, q
25:         route ← route ∪ finish route
26:         i+ = 2
27:         route done = true
28:      else if not time window feasible then
29:         remove target and append it to pending
30:         node ← modified solution(i)
31:         if i + 1 > |modified solution| then
32:            t, d, q, finish route to depot(node, t, d, q
33:            route ← route ∪ finish route
34:            i+ = 1
35:            route done = true
36:         end if
37:      else if not energy feasible then
38:         s ← chose closest feasible station to both node and target
39:         update time and charge
40:      else
41:         route, t, d, q, k ← route costumers(node, target, route, t, q, k, v, r, d, dist)
42:         if i + 1 > |modified solution| then
43:            t, d, q, finish route to depot(node, t, d, q
44:            route ← route ∪ finish route
45:            i+ = 1
46:            route done = true
47:         end if
48:      end if
49:   end while
50: end while
51: return parent, distance, time
```

Algorithm 3. Reconstructing\Repair operator.

The algorithm takes as input the modified solution (one single vector with the permutation) and returns the parent coded in the initial representation exposed in Section 2. The algorithm initializes all the storage variables in Lines 1-5. While there are positions of the modified vector that have not been added to a route (Line 6) the following process will be looped. The modified solution vector is appended with the removed costumers from the past iteration (Lines 7-8). The new route is initialized (Lines 9) and the current position is added to the route updating

all values (Line 12). While the route is not finished yet (Line 14), the target will be the next position of the modified vector (Line 15). Lines 16-20 evaluate if the transition between the current node and the target is feasible in terms of load. If it's not, then the vehicle is routed to the depot and the route is terminated. Lines 21-27 evaluate if the transition between the current node and the target is feasible in terms of the total time. If it's not, then the vehicle the current node is removed from the route, and it's terminated. Lines 28-35 evaluate if the transition between the current node and the target is feasible in terms of the time window of the target. If it's not, then the target is removed from the modified solution and appended to the pending list. Lines 37-39 evaluate if the transition between the current node and the target is feasible in terms of the energy of the vehicle. If it's not, then the vehicle is routed to the nearest station to the current node and the target. Finally, all previous conditions are not satisfied (feasible on all dimensions), then the target is appended to the route (Lines 41-46). Figure 6. showcases how the repair operator would insert a station between costumers 'C3' and 'C4' if there were not enough energy to traverse this arc.

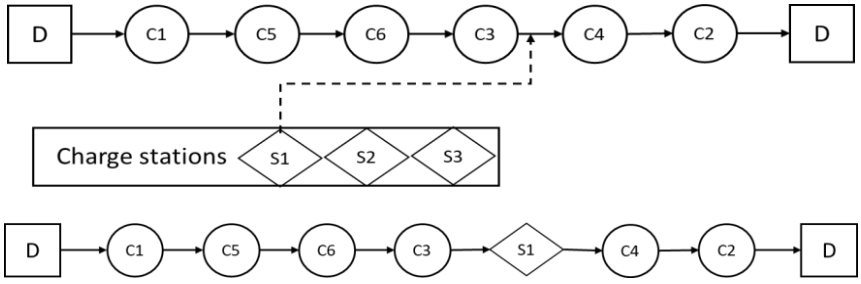


Figure 6. Reconstructing\Repair operator example (energy unfeasible)

4 Computational experiments

4.1 Literature instances

To validate the performance of the proposed heuristic, we used classic instances proposed by (Schneider, Stenger, & Goeke, 2014). Those instances contain a set of 56 large-sized instances consisting of 100 customer locations and 21 charging stations, and a set of 36 small-sized instances consisting of 5, 10, and 15 customer locations. The charging stations in each instance are determined as follows: One of the charging stations is located at the depot node. The remaining charging stations are located randomly with the assumption that each customer can be reached from the depot using at most two charging stations. The VRPTW dataset is divided into three classes based on geographical distribution: random customer distribution (R), clustered customer distribution (C), and a mixture of both R and C classes (RC).

4.2 Technical specifications

Computer specs

The computational experiments were performed on Python 3.10.4 and on two different computers with the following specs:

1) MacBook Pro 2019, CPU 6-Core Intel Core i7 of 2,6 GHz and 16 GB RAM.
2) Dell G15, CPU Ryzen 7 5800H, 4 GHz and 16 GB RAM.

General hyperparameters

These are some of the hyperparameters set over all the experiments.

- RCL $\boldsymbol{\alpha}$: 0,5
- RCL criterion: Hybrid (chosen randomly)
- Crossover rate: 0,5
- Mutation rate: 0,5
- Elite percentage: 5%

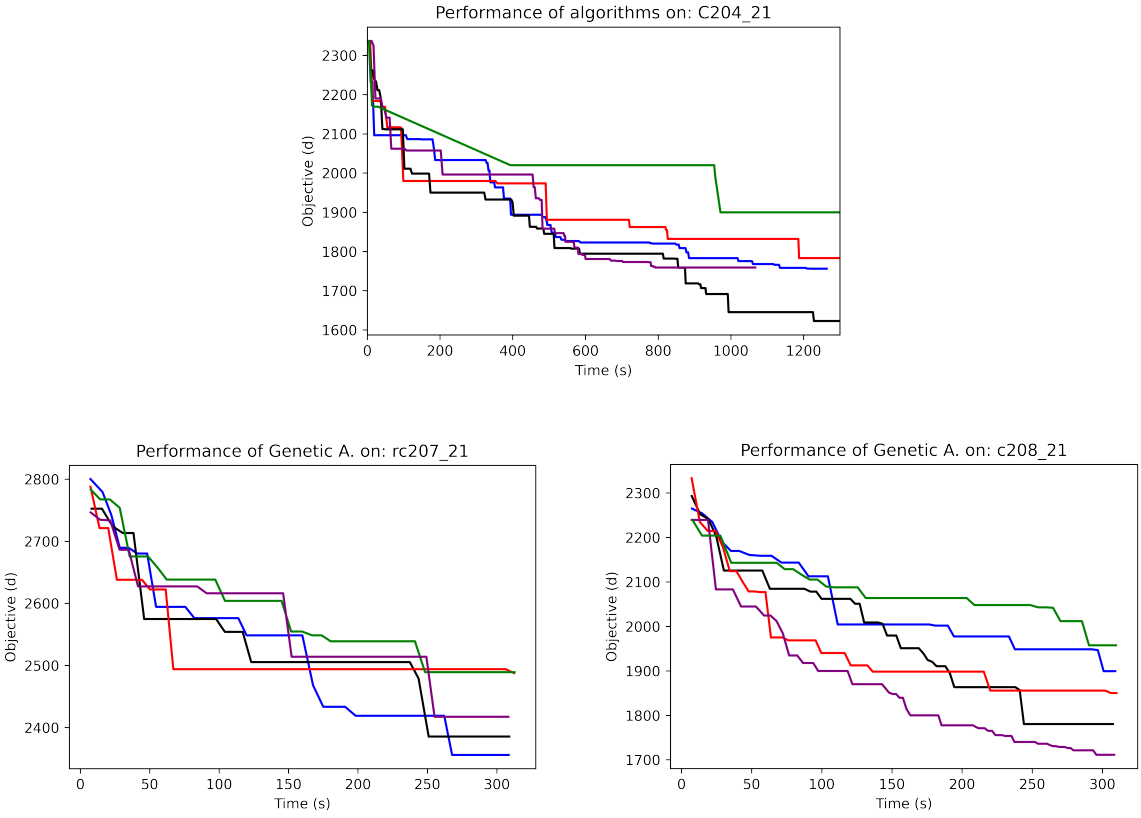
Other parameters were tuned during the experiments and will be indicated opportunely.

5 Results

5.1 Performance

5.1.1 Operators

To evaluate the performance of the operators, we tested them on three of the largest instances. On Graph 1, the operators: simple crossover, 2-Opt, simple insertion, smart crossover, and hybrid, are shown in blue, red, black, purple, and green, respectively. On the top the performance of all operators on instance *c204_21* is displayed. For this instance, the time horizon of 1200 seconds on instance. Each one was run with a maximum number of 400 generations, explaining why some configurations don't reach the whole time-window. There were 500 individuals on each generation. On the bottom-left and bottom-right the performance on instances *rc207_21* and *c208_21* is displayed. On both, all operators were given a time limit of 5 minutes, a population of 750 individuals and no maximum generations.



Graph 1. Performance of all operators on instances: c204_21, rc207_21, c208_21

Observing the performance of the operators on all instances, we considered to continue the research with 2-opt (red), simple insertion (black) and smart crossover (purple).

5.1.2 Instances testbed

Having defined all components of the algorithm, we run the tested of instances compared with the literature best known solutions (Schneider, Stenger, & Goeke, 2014), our constructive for measuring the hybrid genetic algorithm performance.

Inst.	BKS		Constructive		Hybrid Genetic Algorithm			
	# EV	F.O	F.O	Gap	# EV	F.O	t	Gap
101C10	3	393,76	454,96	15,5%	4	375,29	1800,99	**-4,7%**
c101C5	2	257,75	296,1	14,9%	3	250,01	1800,59	**-3,0%**
c101_21	12	1053,83	2867,51	172,1%	21	2352,5	1829,41	123,2%
c102_21	11	1056,47	2557,2	142,1%	21	2069,8	1820,64	95,9%
c104_21	10	979,51	2060,33	110,3%	18	1821,18	1812,34	85,9%
c106_21	11	1057,87	2579,32	143,8%	21	2179,41	1814,72	106,0%
c108_21	10	1100,32	2297,94	108,8%	20	1971,16	1814,88	79,1%
c202_21	4	645,16	2447,05	279,3%	12	1827,15	1812,6	183,2%
c207_21	4	635,17	2191,05	245,0%	12	1885,71	1817,05	196,9%

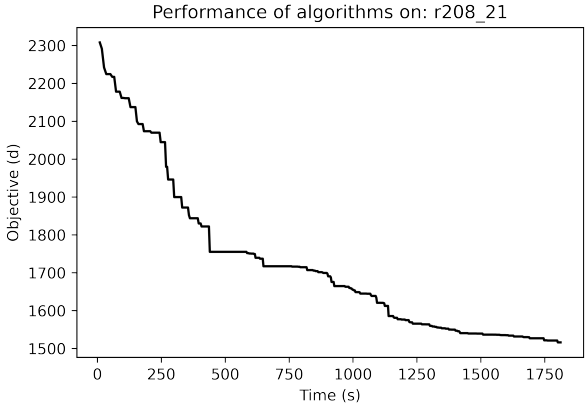
c208C15	2	300,55	449,18	49,5%	2	304,83	1801,11	1,4%
r107_21	12	1154,52	2407,99	108,6%	24	2094,82	1815,39	81,4%
r109_21	12	1294,05	2421,93	87,2%	26	2233,26	1815,08	72,6%
r201_21	3	1264,82	2565,14	102,8%	11	2244,81	1811,75	77,5%
r202C15	2	358	524,5	46,5%	3	398,97	1801,1	11,4%
r202C5	1	128,78	152,59	18,5%	2	143,13	1800,23	11,1%
r204_21	2	790,57	2339,86	196,0%	6	1673,31	1811,21	111,7%
r208_21	2	736,6	2307,93	213,3%	4	1515,88	1812,08	105,8%
rc103C15	4	397,67	485,51	22,1%	5	428,67	1801,33	7,8%
rc105C5	2	241,3	239,46	**-0,8%**	3	239,46	1800,29	**-0,8%**
rc105_21	14	1475,31	2761,75	87,2%	25	2417,47	1818,25	63,9%
rc108C10	3	345,93	452,74	30,9%	4	396,22	1800,93	14,5%
rc201_21	4	1444,94	3221,69	123,0%	14	2905,42	1813,05	101,1%
rc203_21	3	1073,98	2688,04	150,3%	8	2012,92	1812,96	87,4%
rc204C5	1	176,39	187,69	6,4%	1	185,16	1800,31	5,0%
			Avergae	103,05%			Avergae	67,27%

5.2 Evolution

For better insight in the functioning of our algorithm, we captured the evolution of the incumbent throughout the algorithm's process. The following sub-section present this evolution for some instances along with the number of costumers:

5.2.1 *r208_21*

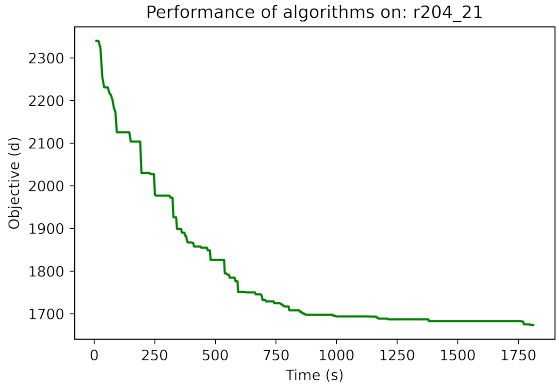
- *Costumers: 100*



Graph 2. Performance of HGA on instance: r208_21

5.2.2 *r204_21*

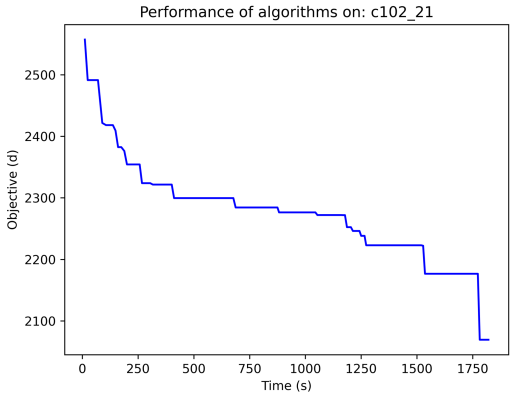
- *Costumers: 100*



Graph 3. Performance of HGA on instance: r204_21

5.2.3 c102_21

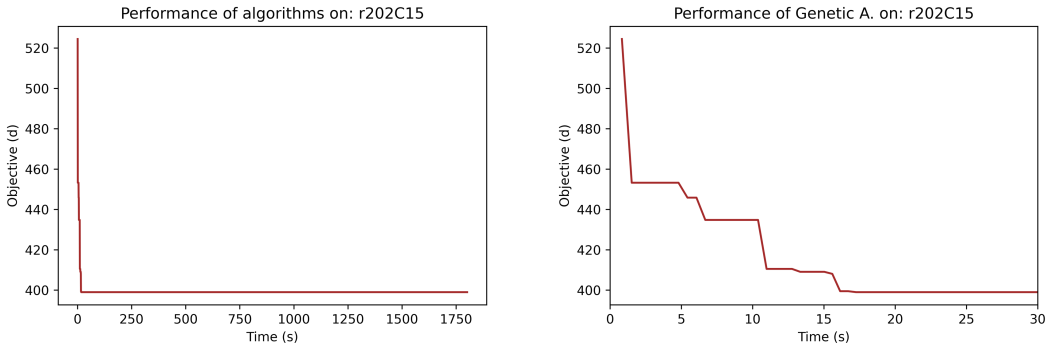
- *Costumers: 100*



Graph 4. Performance of HGA on instance c102_21

5.2.4 *r202C15*

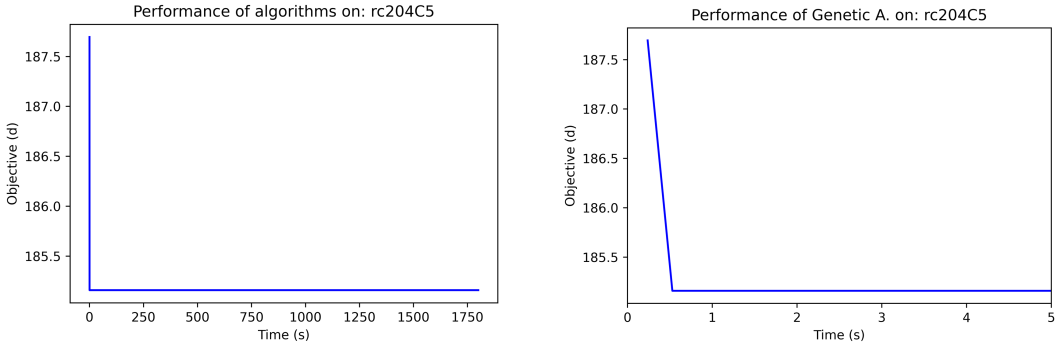
- *Costumers: 15*



Graph 5. Performance of HGA on instance r202C15

5.2.5 *rc204C5*

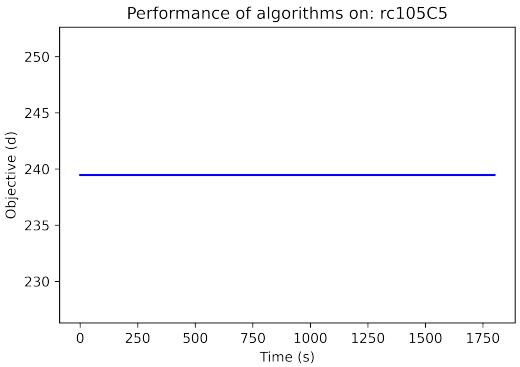
- *Costumers: 5*



Graph 6. Performance of HGA on instance rc204C5

5.2.6 *rc105C5*

- *Costumers: 6*

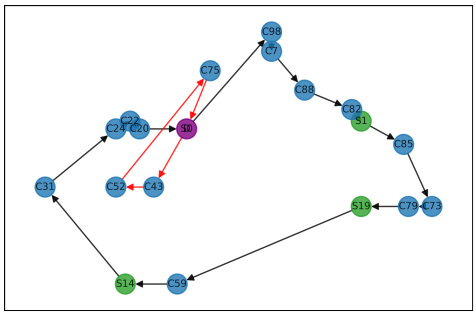


Graph 7. Performance of HGA on instance rc105C5

5.3 Routes

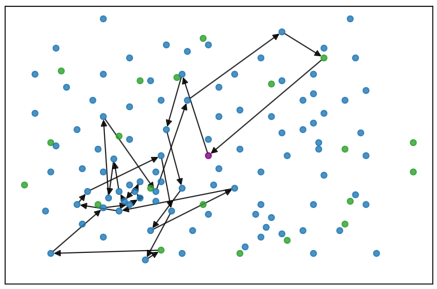
Other tool to gain insight on the performance of the algorithm are the routes. The final solution of the problems are the routes the metaheuristic produces. Ergo, reviewing the routes designed by the operators might reveal areas of work or development.

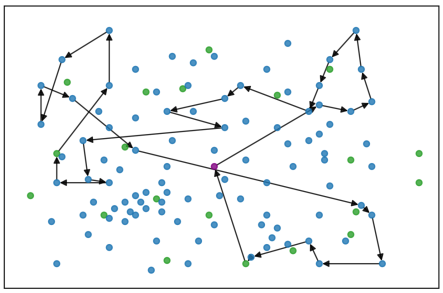
5.3.1 *c208C15:* Generated two routes to service 15 costumers

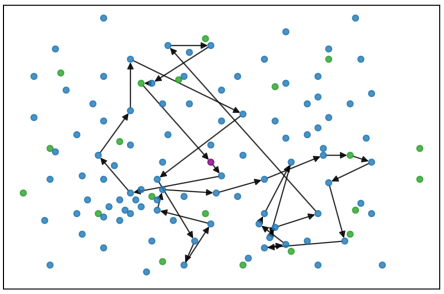


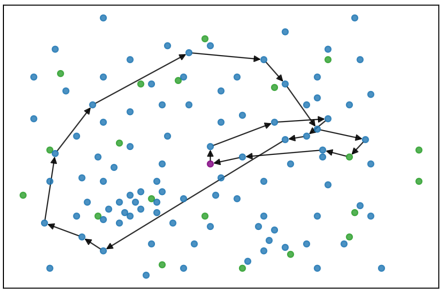
Graph 8. Best generated routes for instance c208C15

5.3.2 *r201_21:* Generated four routes to service 100 costumers



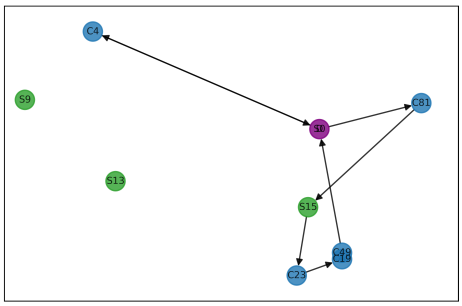






Graph 9. Best generated routes for instance r201_21

5.3.3 *rc204C5:* Generated one route to service five costumers



Graph 10. Best generated routes for instance rc204C5

5.4 Analysis

All the experiments and the information recollected allow to perform a detailed evaluation over the model and the experiment itself. Firstly, the algorithm was available to perform and solve all the instances we tested. All solutions along the evolution process were feasible, we can provide feasible solutions very early on the process.

Firstly, over the different operators Graph 1 shows that some operators are overall better but had a similar behavior. The performance of the different operators can also depend on the size of the instance, leaving room for this analysis on instances varying on sizes. Once the operators of the HGA were defined the results are presented on Table x. Overall, the algorithm averaged 67% from the best-known solution and the constructive heuristic 103%. Even the gap achieved by the HGA are high, looking at some patterns might reveal as why. For small instances, the algorithm has a remarkable performance, achieving in three instances a better solution than the best-known solution. Furthermore, one of those solutions was from the instance in which the evolution process made no improvement on the solution. This means, the constructive built a better solution than the best-known and the genetic algorithm couldn't improve.

On bigger instances however, the gaps are at order of 100-190%. First, the evolution process given for each instance was considerably low. More exploration and recombination time would most probably result in better solutions. Second, this supports the notion given by Graph 9 that routes on bigger instances should be more efficient. More complex operators and designed for bigger routes could be the solution to better solutions. Third, constraining the number of vehicles on the solutions could also avoid local optimum.

Other relevant aspect to consider are the number of vehicles of the optimum solution, which reflects in some way its efficacy. Some of the larger instances have many vehicles (24-26). This can also be studied to determine what can be responsible and what counter measures can be implemented into the algorithm to avoid this high vehicle-quantity solutions. The difficulty of achieving high-quality solutions on bigger instance can also be attributed to the innate nature of the operators of the HGA. The philosophy of the operators is basically to twist a solution vector with itself. This means, limits the search to operations within the same solution and therefore the solution space explored. If between-parents operators were implemented, in which two routes from two different solutions were somehow combined, the exploration would be drastically boosted

Around the evolution process of the algorithm there is a lot of room to comment and question. Graphs 2 and 3 both correspond to big instances (100 costumers) and portrait a similar behavior. Over the first seconds of the process, the algorithm finds substantially better solutions very fast. As time advances, the algorithm reduces the rate at which it finds better solutions and the differential improvement. Finally, towards the second half of the evolution time, the algorithm stabilizes and finds betters solutions slower and slower. However, from the two Graphs, the latter seems as an early stage of the first one. Therefore, it would be expected to keep improving in solutions at a reasonable rate if the time of exploration was extended. This showcases how the running time is also a hyperparameter that must be calibrated. The evolution on another instance with 100 costumers is presented on Graph 4. However, there is an interesting phenomenon with this instance. Until around 1200 seconds it displays a very similar behavior as the two previous instances. At around 1500 and 1750 seconds there are two transitions between generations in which the incumbent is significatively improved. Studying up close what might cause this abrupt local optimum change, can guide the exploration through the solution space and avoid local optimum.

Very similar behavior, on a different scale, can be observed at smaller instances. For example, on the left Graph 5 presents the evolution of the incumbent. On the right, the x-axis is rescaled to 30 seconds. When the number of costumers is reduced, the reduction on the cost appears much earlier and stabilizes under a minute. Smaller instances will certainly require less exploration time than the big ones. Additionally, special operators might be required for smaller instances. For example, let's consider instance *rc204C5* with 5 costumers is Graph 6. The genetic algorithm only achieved one improvement at 0,5 seconds from the start. This evidence the utility of a termination criterion over generation/time without improvement. On Graph 7, for instance *rc105C5* the genetic algorithm provided no improvement whatsoever from the solution of the constructive. For these smaller instances, perhaps such advanced algorithms are unnecessary and the constructive with some variants can provide a high-quality solution without unnecessary complexity.

Regarding the graphs, the routing appears to be reasonable and logical in smaller and intermediate instances as evidenced on Graph 8 and Graph 10. Routes cover nearby nodes and rely on charging stations to avoid excessive number of vehicles. On larger instances such as *r201_21* in Graph 9, the routes appear to traverse all the grid and covering excessive distances on single trajectories. A more intelligent to route the routing could result on more efficient routes that cover costumers on specific parts of the map and reduce the distance. Nevertheless, the time window constraint might explain some of this behavior as the routing is always directed by feasible time windows that might not be close by. The bottom right route of Figure 9 displays a loop-shaped route that showcases how a desired route might look like.

5.5 Additional comments

There are various aspects to remark from this research.

- When comparing and analyzing the performance of the different instances and configurations of the metaheuristic, an aspect to consider is the lack of consistency on the experimentation phase. The use of two computers with different specifications and different levels of use, on other tasks, during the experimentation phase add noise to the computational metrics. Additionally, experiments were run in parallel in batches under inconsistent conditions (we are master and doctorate students with various other responsibilities). Therefore, any analysis of performance across different instances can be affected and distorted.
- The constructive and repair operator both share somewhat the same philosophy and even the same code methods. Therefore, the achieved results all are heavily influenced by these basic operators such as a '*route to depot*'. It is possible that some regions of the solution space aren't being explored because of this logic. For example, when a vehicle has not enough charge, it reaches the closer station to charge. However, the best station might be the one closest to the best target candidate. Or, when a vehicle has full capacity, it must end the route. However, if the vehicle directs to the depot, resets the charge, and leave again, it might create more efficient routes as for clients with late time windows.
- Along all this work, feasibility on all individuals of a given population is guaranteed. This is a great contribution, delivering feasible solutions in a reduced time. However, this also poses a research opportunity. By saving some good unfeasible solutions, promisor neighborhoods can be explored from those solutions.

6 Conclusions

We propose a genetic search metaheuristic to new practical problems in literature for electrical vehicles capacitated vehicle routing problem with time windows. The space search is based on feasible routes; therefore it is possible to decompose the problem into assignment problem and sequencing the costumers or charging stations to verify feasibility.

The paper introduces several methodological contributions. In the crossover and mutation operators, the management of only feasible solutions, the individual evaluation procedure driven both by solution cost and the contribution to

feasible population diversity, more generally, the adaptive population mechanism that enhances diversity, allows a more extensive access to reproduction, and preserves the good solutions represented by the elite individuals.

The proposed solution approach was tested on the small, medium, and large-size datasets. We also use labeling and greedy evaluation methods to insert recharging stations or slices routes to minimize distances and fulfill demands at time windows. The different operators and configuration of the metaheuristic were also tested.

7 Future work

The solution method provides feasible and improved solutions over a simple constructive heuristic. Nevertheless, there are several areas of opportunity to advance this work to achieve higher quality solutions and increase the applicability of the product. These include:

- Extend the number and complexity of the operators and phases of the genetic algorithm. More variety of operators increases the capacity of the metaheuristic to escape from local optimum and explore faster and more efficiently the solution space. Furthermore, a learning procedure could be implemented to calibrate the use of the different operators in function of the characteristics of the instance (number of nodes, energy capacity, vehicle capacity, etc.).
- Consolidate the testing performed during this phase of the project. Some instances were not solved, in others the algorithm could have performed better with more time. A much rigorous process to determine the hyperparameters of the algorithm could have had tremendous impact on the quality and times of the solution.
- In the present work the electrical vehicle must fully recharge the battery when it arrives at a charging station, costumer's time windows could be exploited more efficiently if partial recharge on EVs is considered. The objective could be focused on minimizing the waiting time at stations and covering the costumers and different customer requests and variety of time windows of customers.
- In the present work the demand of the electric vehicle is known beforehand. However, a more realistic, and of interest to modern delivery companies, scenario would be to have requests arrive dynamically through the time horizon as well as stochastically. Another variation to consider would be stochasticity on the travel times between two nodes of the network. Considering all of this, a general framework for stochastic optimization could be implemented by considering a sequential decision process.

8. Bibliography

Cordeau, J. F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 928-936.

Cordeau, J. F., Laporte, G., Savelsbergh, M. W., & Vigo, D. (2007). Vehicle routing. In *Handbooks in operations research and management science* (pp. 367-428.).

Gendreau, M., & Potvin, J. Y. (2018). *Handbook of Metaheuristics.* Berlin/Heidelberg: Germany: Springer.

Li, H., & Lim, A. (2003). Local search with annealing-like restarts to solve the VRPTW. *European journal of operational research*, 115-127.

Pelletier, S., Jabali, O., & Laporte, G. (2016). 50th anniversary invited article—goods distribution with electric vehicles: review and research perspectives. *Transportation science*, 3-22.

Schneider, M., Stenger, A., & Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation science*, 500-520.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operation research*, 611-624.