A two-step optimization approach for a stochastic multi-stage capacitated vehicle routing problem

Author: Chiara Vercellino

Supervisor: Prof. Paolo Brandimarte

Master Degree in
Mathematical Engineering

Department of Mathematical Sciences
Politecnico di Torino
Italia
A.Y. 2019/2020

Contents

List of Figures

List of Tables

Introduction

Nowadays we benefit from the application of optimization algorithms in several fields, one of them is the logistic field in which *Vehicle Routing Problem*, in short *VRP*, settles down: *VRP* aims to find optimal routes, given a fleet of vehicles that start from a depot and deliver a certain good to a set of customers.
Starting from the first definition of *VRP*, lots of variants were designed to deal with the situations faced in the real-world: the subject of this thesis is *multi-period Capacitated Vehicle Routing Problem* with stochastic customers' demands over time.

The study and the definition of this problem come from the real-world problem of a furniture's delivery company: each day a set of stochastic customers shows up and the company has to decide which customers to serve in the following day, according to the deterministic pending demands up to that day and stochastic information about futures. After that, convenient routes have to be found to schedule the path of each vehicle used to serve the daily customers.
Currently, there are a lot of solvers for *VRP* and its deterministic variants, some of them are open source, like *OR-Tools* and *CPLEX*, others are commercial software, like *Gurobi*. Each of these solvers has its own approach to the optimization problem and allows the user to set different kinds of constraints and decision variables. Anyway, they can be used to solve only the deterministic problem and do not take into account the stochastic information, since it cannot be modelled into them.

Starting from this observation, the first task of this thesis is to find a policy to choose the set of customers to serve each day in order to minimize the total travel and service cost in the long run, exploiting the stochastic information. The set of the selected customer is then passed to *OR-Tools* solver to find feasible and convenient routes.

Given an improving policy, with respect to naïve policies, the second part of this thesis consists in the implementation of a meta-heuristic, based on *Tabu Search* approach and *Clark-Wright* algorithm, to improve the convenience of the daily routes. The previous results obtained with *OR-Tools* solver are considered as benchmarks.

Chapter 1

Dynamic Capacitated Vehicle Routing Problem

This chapter contains a description of the mathematical optimization model for the *multi-period Dynamic Capacitated Vehicle Routing Problem* (*multi-period DCVRP*). It is a multi-period and multi-stage problem, with a daily period. The stochastic part of the problem, which makes it *Dynamic*, concerns the customers' demands because future demands are uncertain: there is not a set of fixed customers, the demands in terms of kg of requested furniture, the service times and the positions of future customers are unknown. What is known is the probability distribution of customers in the region of interest.

Although the optimization problem can be mathematically defined with only one model, the chosen optimization approach consists of two main steps that are repeated day-by-day:

- **Customers' selection**: select the most convenient customers to serve, in this step the capacity constraint of available vehicles and the stochastic information about future demands have to be taken into account.
- **Route Plan**: given the selected customers, find the less costly routes for deliveries. The cost of the routes has both contributions of travel time and of the number of used vehicles. The shorter the routes the more convenient is the solution, also the number of used vehicles has to be minimized.

This division of the problem in two steps matches with the two main tasks of this thesis.

1.1 From VRP to multi-period DCVRP

To describe the optimization problem, we start from the most basic version of it, the *Vehicle Routing Problem*. Then, we specify the characteristics of more sophisticated versions, according to the case study problem.

1.1.1 Vehicle Routing Problem

The *Vehicle Routing Problem* (*VRP*) is an extension of the *Travel Salesman Problem TSP*. Indeed, *TSP* consists of finding the optimal path to visit a set of customers, starting from one of them and then coming back to the starting customer, all other customers are visited exactly once.
In *VRP* we still have a set of customers that we want to visit exactly once, but the starting point is a depot, from which a fleet of vehicles starts: each vehicle starts and ends its route in the depot. *TSP* is simply *VRP* with only one vehicle and the depot coinciding with the starting customer.

The introduction of more than one vehicle implies that the routes of each vehicle must be non-overlapping: each vehicle visit a subset of the original customers and the visiting order is optimized in *TSP* mode.
This means that a possible decomposition of the *VRP* could be to first generate clusters of customers visited by each vehicle and then to optimize the routes. The optimization step should consider each vehicle's route problem as a *TSP*, starting and ending in the depot. Anyway, many other approaches can be followed to solve *VRP*.
Another main aspect of *VRP* concerns the number of vehicles used to solve the problem. In the simplest version of it, the number of available vehicles could be unlimited, but in our definition, the number of vehicles itself defines the feasibility of the solution: we have a limited number of vehicles and we want to find a solution that exploits at most all of them.

1.1.2 Capacitated Vehicle Routing Problem

Given the *VRP*, other constraints can be added according to the real case needs. If we consider constraints on the capacity, *VRP* becomes a *Capacitated Vehicle Routing Problem* (*CVRP*).
The capacity constraints regard the quantity that each vehicle can carry. This kind of constraint can address more than one measure for each vehicle: in the most basic version of *CVRP*, we have only one capacity constraint for each vehicle that models the total customers' demand that a vehicle could deliver.

In our case, the *CVRP* will have:

- A capacity constraint, expressed in *kg*, that sets the maximum load of customers' demands delivered by each vehicle;
- A customers-capacity constraint, regarding the maximum number of customers that each vehicle could serve;
- A time-capacity constraint, expressed in *min*, that specifies the maximum travel and service time for each vehicle.

It must be noticed that vehicles could have heterogeneous capacities constraints, but we will deal only with homogeneous vehicles.

1.1.3 Multi-period Dynamic Capacitated Vehicle Routing Problem

The *CVRP* can be extended to a multi-period version if we consider the problem of solving it multiple times, with different sets of customers each time. If the selection of the customers to serve and the corresponding route optimization is made in different periods, the problem is also multi-stage.
In our problem, the period coincides with the stage step and it is of one day: so each day, we solve a *CVRP* with a certain set of customers.

Till now, the problem has only deterministic aspects, but in our application, it includes uncertainty that makes it a *multi-period Dynamic Capacitated Vehicle Routing Problem* (*multi-period DCVRP*).
The kind of uncertainty we consider regards customers' future demands. In fact, each day we have a fixed depot position, a fixed number of available vehicles (with deterministic capacity) and the customers' demands till that day, but we have to decide which customers to serve. To do this, we optimize the objective function over all the periods and exploit the stochastic distribution of future customers' demands.
It must be observed that, if the customers' demand were deterministic, the problem would become a multi-period optimization problem in one stage since the sets of customers to serve day-by-day could be decided at first.
Having to deal with a stochastic multi-stage problem it is much more difficult: the selection of the daily served customers can be based only on weak future information. The customers served in the previous day cannot be 'unserved' and the daily customers' demands cannot be procrastinated for too many days, hoping that some other near customers will appear: in fact, for each customer we have a stochastic period (expressed in days) to perform the delivery.

1.1.4 The mathematical formulation of multi-period DCVRP

Starting from the *multi-period DCVRP* in [8], we define our optimization problem as a planning problem on a certain day t.
We assume to have a planning time horizon $T = \{1, \dots, r\}$, at day t there is an updated time horizon $T^{'} = \{t, \dots, r\}$ that represents the time interval in which the demands are set.
With set $N = \{1, \dots, n\}$ we represent the known but not yet served customers, that we will call pending customers, this set is updated each day. The overall set of locations is $N_0 = \{0\} \cup N$, where index 0 indicates the depot.
We denote with A the set of all arcs connecting each pair in N_0, so $c_{i,j}$, $\forall (i,j) \in A$, is the travel time cost associated with each arc (i,j).
For each pending customer $i \in N$ we consider a service time d_i and a demand q_i. Each customer can be served in a set of days $\{a_i, \dots, b_i\}$, to make this set feasible, the first day has to become $a_i^{'} = max\{a_i, t\}$, so the set of feasible days for customer i is $\{a_i^{'}, \dots, b_i\}$.
The fleet of available vehicles is denoted by $K = \{1, \dots, m\}$, each vehicle has a capacity Q, it can travel along a route with duration limit D and it can deliver

up to C pending customers. A fixed cost f is associated with each vehicle.

The optimization problem is solved with respect to binary decision variables x^t_{ijkl} and y^t_{kl} defined as follow:

$$y^t_{kl} = \begin{cases} 1 & \text{if vehicle } k \text{ is used for the deliveries on day } l \\ 0 & \text{otherwise} \end{cases}$$

$$x^t_{ijkl} = \begin{cases} 1 & \text{if vehicle } k \text{ travels from } i \text{ to } j \text{ on day } l \\ 0 & \text{otherwise} \end{cases}$$

We observe that the whole formulation of the optimization problem would include the sum over all days in T, with a probabilistic distribution for the customers' demands, anyway we can decompose the whole optimization problem into daily optimization problems, because even if the *multi-period DCVRP* is dynamic, the routing problem solved each day over the planning horizon $T^{'}$ is static, in fact it is based on known orders and routes are decided entirely before their execution.
So the optimization problem at day t is the following:

$$\underset{x^t_{ijkl},\, y^t_{kl}}{\text{minimize}} \quad \sum_{l\in T'} \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x^t_{ijkl} + \sum_{l\in T'} \sum_{k\in K} f y^t_{kl} \tag{1.1a}$$

$$\text{subject to} \quad \sum_{l\in\{a_i^{'},\dots,b_i\}} \sum_{k\in K} \sum_{j:(i,j)\in A} x^t_{ijkl} = 1 \qquad \forall i \in N, \tag{1.1b}$$

$$\sum_{i\in N} \sum_{j:(i,j)\in A} q_i x^t_{ijkl} \le Q \qquad \forall k \in K, l \in T^{'}, \tag{1.1c}$$

$$\sum_{i\in N} \sum_{j:(i,j)\in A} (c_{ij} + d_i) x^t_{ijkl} \le D \qquad \forall k \in K, l \in T^{'}, \tag{1.1d}$$

$$\sum_{(i,j)\in A} x^t_{ijkl} - 1 \le C \qquad \forall k \in K, l \in T^{'}, \tag{1.1e}$$

$$\sum_{(i,j)\in A} x^t_{ijkl} - (C+1) y^t_{kl} \le 0 \qquad \forall k \in K, l \in T^{'}, \tag{1.1f}$$

$$\sum_{j\in N_0} x^t_{0jkl} = 1 \qquad \forall k \in K,\ l \in T^{'}, \tag{1.1g}$$

$$\sum_{i\in N_0} x^t_{i0kl} = 1 \qquad \forall k \in K, l \in T^{'}, \tag{1.1h}$$

$$\sum_{i:(i,h)\in A} x^t_{ihkl} - \sum_{j:(h,j)\in A} x^t_{hjkl} = 0 \qquad \forall h \in N, k \in K, l \in T^{'} \tag{1.1i}$$

$$x^t_{ijkl} \in \{0,1\} \qquad \forall (i,j) \in A,\ k \in K\ , l \in T^{'}$$

$$y^t_{kl} \in \{0,1\} \qquad \forall k \in K\ , l \in T^{'}$$

Looking at this model, it can be noticed that the objective function (1.1a) minimizes both the travel time and the number of vehicles used in the routing plan.

Constraint (1.1b) ensures that each customer is visited exactly once, during its feasible days and only by one vehicle. Constraints (1.1c), (1.1d), (1.1e) are capacity constraints, respectively for the amount of goods that a vehicle can transport, for the working time and for the number of customers visited in the route.
Constraint (1.1f) links the decision variables x^t_{ijkl} to y^t_{kl}, allowing the fixed costs to be associated only with vehicles actually used for the deliveries.
Constraints (1.1g) and (1.1h) state that each vehicle must start and end its route in the depot, the sum over N_0 allows also empty routes, since $c_{00} = 0$.
Finally, constraint (1.1i) is a mass-balance equation that makes the flow conserve through the route: the number of vehicles that come to a customer's location is equal to the number that leave it.

The complexity of this optimization classifies it as an NP-hard problem since it is an extension of *TSP*, which means that its solution requires exact algorithms that have exponential complexity in the worst case.
Generally, the complexity of *CVRP* is not computed, but, if we consider an exhaustive search on the whole solution space, we can compute its dimension. In fact, in each daily problem, we decide which customers to serve among the ones in N; let us consider to visit $p \geq 1$ customers with at most m vehicles, $1 \leq m \leq p$. Then the number of all possible assignments of customers to vehicles is

$$\sum_{i=1}^{m} \binom{p+i-1}{i-1} = \frac{m}{p+1} \binom{p+m}{m}$$

Then, for each cluster of customers that corresponds to a route, we have to consider all possible permutations that start and end in the depot, so if the customers in the cluster are s, the solution space will contain $\frac{s!}{2}$ permutations since we consider symmetric distances and therefore clock-wise and anti-clockwise tours are equivalent. However, if we consider the customers-capacity constraint, we can set an upper bound for the permutation size, such that for each cluster we can have at most $\frac{C!}{2}$ permutations.
So, we can obtain the dimension of the solution space as $\mathcal{O}(\binom{p+m}{m} C!)$.

1.2 The application to the case study

Concerning the *multi-period DCVRP* based on the application to the case study of the furniture delivery company, we can specify some characteristics:

- the planning horizon T is made of $r = 100$ days;
- every day, there are at most $m = 50$ vehicles available, these vehicles are homogeneous with customer-capacity $C = 5$, duration limit $D = 480\ min$, and $Q = 1000\ kg$;
- the fixed cost f is difficult to estimate and to compare to variable costs c_{ij}, anyway, we consider it as such greater than the variable costs that a solution with fewer vehicles is always better.

Everyday a set of new customers show up, these customers' locations are sampled on a rectangular region of $205km \times 215km$. This region is divided into

square subregions, which we will call cells, of dimension $5km \times 5km$. A multinomial distribution is modelled on the region, such that each cell is associated with a probability and the sum of all cells' probabilities sum up to 1. The idea behind this choice is that cells with higher probability will have a higher number of customers' demands.
To be more precise, each day we simulate customers' demands sampling the total number of new orders by an integer uniform distribution $\sim U([175, 215])$, then the association of orders to the cells is made by means of the multinomial distribution. Finally, for each order that belongs to a specific cell, we sample the location in terms of Cartesian coordinates x and y uniformly on the cell.
The multinomial distribution is taken from the previous thesis [7] on the same topic, a representative heatmap of this distribution is reported in figure 1.1.
The position of the depot is central, at coordinates $(102.5,\ 107.5)\ km$.

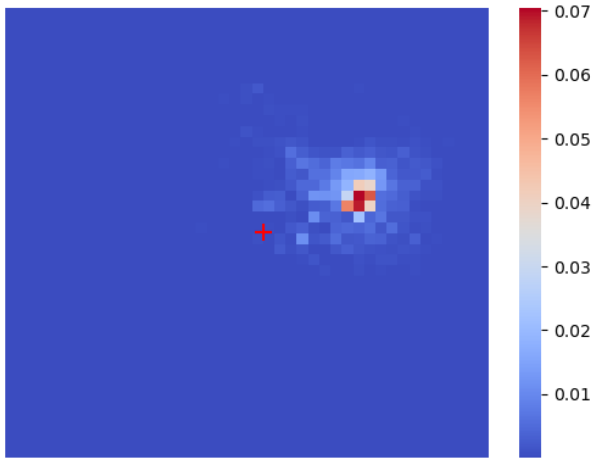


Figure 1.1: Heatmap of the density distribution of customers in the considered region. A high-density area can be noticed in the middle-right part of the figure, the depot is quite near to the high-density area and it is represented by the red cross.

Moreover, the uncertainty of new customers' demand regards also the amount of goods they require and the daily feasibility of their orders.
So, for each new customer, we can have with equal probability, by means of a Bernoulli distribution, a small or a big order: small orders are characterized by light weights $\sim U([5, 10])$ and short service times $\sim U([15, 45])$, on the other hand, big orders present high weights $\sim U([195, 490])$ and long service times $\sim U([45, 135])$. By coherence with the previous units of measurement, service times are expressed in min and weight loads in kg.
Concerning the feasible days, when an order is received at day t, it can be executed in the set of days $\{a_i, \ldots, b_i\}$, with $a_i = t$ and $b_i \sim U([t + 3, t + 5])$.
Still, for the sake of measures coherence, we assume an average speed of vehi-

cles of 60 km/h, in this way we can convert distances between all locations into travel times, so 1 min of travel time corresponds to 1 km of distance.

Finally, in the optimization problem definition, we have assumed that all pending customers can be served in their feasible days period, anyway, this is not always true in our setting, but it depends on the customers' policy selection we use to select the daily customers. If some customers cannot be served within their last feasible days, those orders are not lost, but they become postponed orders, which means that the last feasible day is incremented by 1. This results in the evaluation of another metric that is not present in the objective function (1.1a), to establish the performance of a policy. In fact, we would prefer a solution with less, ideally 0, postponed orders.

Chapter 2

Policies for customers selection

In this chapter. we define policies for the **customers' selection** task, we introduce two naïve policies, *Early Policy* and *Delayed Policy*, whose post-optimization results are considered as benchmarks. This task aims to find a policy that leads to an improvement with respect to benchmark policies, which means, lower travel costs, fewer vehicles used and fewer postponed orders.
To reach this target, two versions of a more sophisticated policy have been implemented, the one called *Neighbourhood Policy.* Analysing the results, obtained with different datasets, the best policy for the problem is the second version of the *Neighbourhood Policy.*

The main characteristics of this policy concern:

- Customers to cells aggregation: to evaluate the convenience of including a customer in the set of the daily selected customers, we need to define his neighbourhood. However, a neighbourhood defined over all possible positions of customers in our region would lead to an uncountable set of possible neighbours, so we need to discretize the region. To do this, the partition in cells has been exploited: each customer is associated with the cell from which he is sampled and the neighbours are defined in terms of cells' center positions.

- Actual vs Expected savings: Given the neighbourhood of a customer, the choice of selecting him for the daily deliveries is associated with the presence/absence of neighbours that make his delivery convenient to be included. The selection of the customer depends on a score that has both contributions of actual neighbours among pending customers and expected future neighbours estimated by the probability distribution on the considered region.

2.1 Policies definition

In the following subsections, we define the policies implemented and applied to the *multi-period DCVRP*.
The *Early Policy* and the *Delayed Policy* are the naïve policies, they were both presented in [7] and [1], but the notation used in this chapter comes from [1]. Also, the first version of *Neighbourhood Policy* takes inspiration from the paper cited above, but some important modifications were introduced to fit the specifications of the problem.

All these policies are meant to be applied day by day and to lead to a feasible solution of the daily *CVRP*. To guarantee the feasibility, the following algorithm has been applied.

Algorithm 1 Customer Selection

```
1: Choose the policy
2: for $t \in 1, ..., T$ do
3:     $N \leftarrow$ set of pending customers
4:     $S \leftarrow$ set of selected customers based on policy
5:     $solution \leftarrow False$
6:     while not $solution$ do
7:         Try and solve $CVRP$, if it is feasible: $solution \leftarrow True$
8:         if $solution == False$ then
9:             Remove last customer in $S$
```

To avoid wasting too much execution time in the **while** cycle an upper bound k_{max} on the number of selected customers in S is computed starting from the aggregate capacity of all available vehicles. In particular, given the set of pending customers N and m available vehicles:

$avg\_q = average\{q_i\}$, $i \in N$ is the average load

$avg\_d = average\{d_i\}$, $i \in N$ is the average service time

$\widetilde{Q} = m \times Q$ is the aggregate load capacity

$\widetilde{D} = m \times D \times perc$ is the aggregate duration capacity for service time

Then the upper bound is the integer approximation of

$$k_{max} = min\left(\frac{\widetilde{Q}}{avg\_q}, \frac{\widetilde{D}}{avg\_d}\right)$$

The value of *perc* is fit to the data, with our dataset, a suitable value has been found out as $perc = 0.67$. This means that, on average, more than half of the available working time of each vehicle is reserved for the service task.

2.1.1 Early Policy

The *Early Policy* (*EP*) consists of selecting, among the pending customers of a certain day, all possible ones. If not all customers fit the capacity constraints,

then as many as possible customers are selected according to an urgency criterion.
In particular, the priority goes to orders

- that have yet been postponed: customers who see their orders postponed will be unsatisfied, this priority rule is made to avoid delivery delays greater than one day;
- with expiration day closer to the current day: to reduce as much as possible the postponed orders.

If two or more orders have the same expiration day, the priority goes to the yet postponed orders.
The main idea behind the *EP* is to reduce the waiting time for each pending customer. This can result in a very convenient policy for customers and could be a smart policy even for the company if the customers' demand distribution was quite uniform on the region or inventory costs were such high to encourage a fast clean-out.

2.1.2 Delayed Policy

The *Delayed Policy* (*DP*) consists of selecting, from the pending orders, only the ones whose expiration days coincide with the considered day. Also, in this case, a priority criterion that privileges orders that have yet been postponed is set.
It must be noticed that *DP* has a higher probability to produce postponed orders with respect to *EP*, since if some of the pending customers, that should be selected in the current day, do not fit in the available vehicles, their orders will for sure be postponed.
This kind of policy can be convenient in case of few daily orders, so the pending orders are delayed as much as possible, hoping that future orders would exploit better the capacity of the available vehicles.

2.1.3 Neighbourhood Policy: first version

The first version of *Neighbourhood Policy* (*NP*) is based on the concept of customers to cells aggregation. In fact, the association of each customer to the corresponding cell allows the computation of estimated savings indexes between pairs of customers.
The saving index I_{ij}, between customers $i, j \in N$, is a common measure for routing problems and it expresses the potential savings derived from serving both customers using the same route:

$$I_{ij} = \frac{c_{0i} + c_{0j} - c_{ij}}{2(c_{0i} + c_{0j})} \qquad i, j \in N$$

As can be seen in the equation above, the more the customers i and j are near, the more the savings index is closer to 0.5, $I_{ij} = 0.5$ iff $c_{ij} = 0$. In general, the higher the value of the saving index, the more convenient is to put both customers on the same route.
Anyway, it is not convenient to use this index computing directly costs on customers' locations because, on one hand, we do not know the locations of future

customers and on the other hand the computation of the distance matrix between all pairs of n pending customers is costly, it scales as $\mathcal{O}(n^2)$ and since not all the pending customers will be selected the computation of all these distances could be partially needless.
These issues are overcome through the associations of customers to the cells: in the considered region, we have a fixed set of cells $V = \{1, \dots, v\}$, each cell has a center, so we can consider as an approximate position of customers the center of the cell to which they are belonging. The depot position 0 is not approximated to the center of a cell because it is fixed and it will be present for all the days in the simulation.
So, after having computed the pairs distances $\widetilde{c}_{ij}$, with $i, j \in V_0 = \{0\} \cup V$, the approximated saving index becomes:

$$\widetilde{I}_{ij} = \frac{\widetilde{c}_{0i} + \widetilde{c}_{0j} - \widetilde{c}_{ij}}{2(\widetilde{c}_{0i} + \widetilde{c}_{0j})} \qquad i, j \in V$$

Since the cells are squared $5km \times 5km$, the approximation error between a pair of customers is at most $2\sqrt{2.5^2 + 2.5^2} \approx 7km$. These indexes are computed only once at the beginning of the simulation.
Given $\widetilde{I}_{ij}, \forall i, j \in V$, it is possible to compute the set of neighbours cells for a certain cell $i \in V$:

$$V_\rho(i) = \{j \in V : \widetilde{I}_{ij} \geq \rho\}$$

The parameter ρ, to be tuned, controls the dimensions of the neighbourhood, higher values of ρ lead to a smaller neighbourhood: from the previous observation, we know that $\widetilde{I}_{ij} \leq 0.5$ and by triangular inequality $\widetilde{I}_{ij} > 0$, this comes in handy to set a range for the tuning phase, $\rho \in (0,\ 0.5]$. If $\rho = 0.5$ only customers belonging to the same cell of customer i form his neighbourhood.

At a certain day $t \in T$, the set V is partitioned in the set of active cells V^t, i.e. the cells that are associated with locations of pending customers, and in the set of inactive cells $V \setminus V^t$. Then, we estimate the convenience of serving customer i at time t computing index $conv_i^t$, according to the following considerations:

- customers with very few available service days may be very urgent to serve;
- we know the one-step probability p_{ij}^t that a customer in cell j in the neighbourhood of cell i will require service the following day. That is simply the probability associated to cell j in the multinomial distribution;
- it may be convenient to postpone customer j if p_{ij}^t is sufficiently large;
- the priority concerning yet postponed orders and the expiring ones must be maintained.

These observations are summed up into the definition of $conv_i^t$:

$$conv_i^t = \begin{cases} \dfrac{1}{b_i - t}\left(1 + \dfrac{1}{|V_\rho(i) \setminus V^t|} \sum\limits_{j \in V_\rho(i) \setminus V^t} (1 - p_{ij}^t)\right) & b_i > t, V_\rho(i) \setminus V^t \neq \emptyset \\ \dfrac{M}{b_i - t} & b_i > t, V_\rho(i) \setminus V^t = \emptyset \\ M & b_i = t, \text{ not yet postponed} \\ M + \gamma & b_i = t, \text{ yet postponed} \end{cases} \tag{2.1}$$

where b_i is the last available day for order i, M is a parameter that guarantees that whenever possible, all orders that reached their expiration date will be selected at day t, and γ is a reinforcing term that gives the highest priority to yet-postponed orders.

Finally, the customers' selection at day t is performed by sorting the customers by decreasing $conv_i^t$, and taking the top k ones with

$$k = min(k_{max},\ k_\alpha)$$
$$k_\alpha = |\{i \in N : conv_i^t \geq \alpha\}|$$

It may seem that *NP* introduces a lot of parameters to be tuned, but actually, the parameters to tune are only three: ρ, M and α, moreover the range for the tuning of these parameters can be estimated. In fact, γ can be arbitrary set to a value greater than 0, as it is only needed to give the priority to yet postponed orders, among expiring orders, so we set $\gamma = 1$. The range for M can be estimated looking at some bounds.
In the definition of $conv_i^t$, the term

$$1 + \frac{1}{|V_\rho(i) \setminus V^t|} \sum_{j \in V_\rho(i) \setminus V^t} (1 - p_{ij}^t)$$

has value 2 as upper bound since $p_{ij}^t > 0$, whereas its lower bound is 1, this lower bound can be reached in the limit case $V = V_\rho(i) \setminus V^t$.
Then, considering that, by definition, in our dataset $1 \leq b_i - t \leq 5$

$$\frac{1}{5} \leq \frac{1}{b_i - t} \left(1 + \frac{1}{|V_\rho(i) \setminus V^t|} \sum_{j \in V_\rho(i) \setminus V^t} (1 - p_{ij}^t) \right) < 2$$

Looking at the second equation in (2.1), which concerns customers that have all active neighbours, we can see that, if we set the value of $M \geq 10$, those customers will be selected independently from their time availability. This situation must be avoided because the customers-capacity of vehicles is limited and with high probability, the kind of customers that could lead to a situation in which $V_\rho(i) \setminus V^t = \emptyset$ are the customers belonging to high-density cells; these customers are also quite near to the depot and they can be easily visited by routes going to more distant customers in low-density cells. So, if we select too many customers in high-density cells we risk to waste capacity and do not be able to exploit the presence of clusters of near customers in low-density cells.

Following these considerations, we can analyse the behaviour of $\frac{M}{b_i - t}$ to further reduce the range for optimal M value. Looking at figure (2.1), considering customer i such that $V_\rho(i) \setminus V^t = \emptyset$, we can see that if we set $M = 8, 9$, he will be selected anyway, even if the number of available days to serve him is ≥ 4; with $M = 6, 7$ the minimum value of $b_i - t$ that leads to a sure selection of customer i is 3. $M = 2, 3$ makes customer i to be for sure selected only if $b_i - t = 1$. Knowing that $b_i \sim U([t + 3, t + 5])$, we decided to look for values of M that guarantees the selection of customer i when $b_i - t$ is at least 2, so the choice of M has to be made in the range $[4, 5]$.

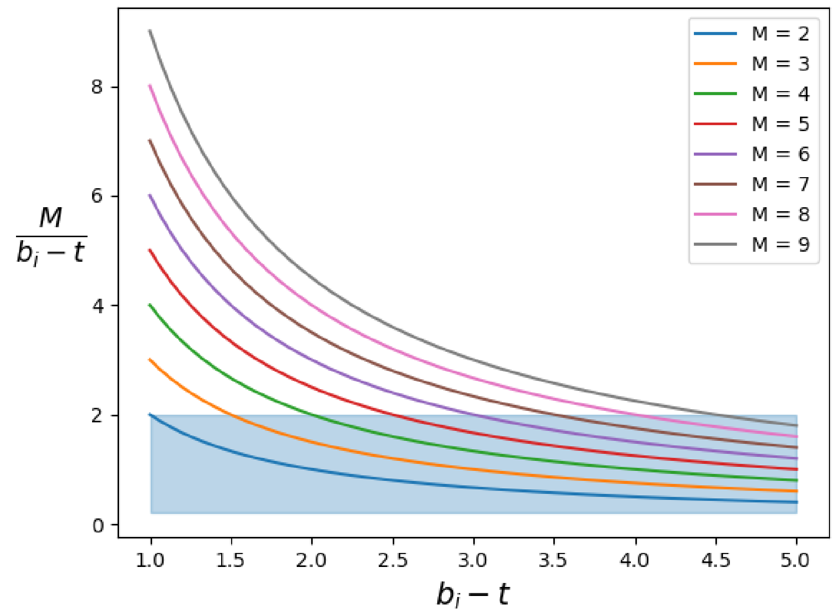


Figure 2.1: Values of $\frac{M}{b_i - t}$ for different values of M, the blue region highlights the region $(\frac{1}{5}, 2)$, that is the range in which we want to set most of the indexes $conv_i^t$, for $b_i - t \in \{1, 2, 3, 4, 5\}$

.

Concerning the parameter ρ, as said above an adequate interval for tuning is $(0, \frac{1}{2}]$, whereas α must be chosen in $(\frac{1}{5}, 2)$.

To sum up, the following algorithm reports the main steps of *NP* implementation.

Algorithm 2 *NP* for customers selection

```
1:  Compute $\widetilde{I}_{ij} \quad \forall i, j \in V$
2:  Set a value for $\rho \in (0,\ 0.5]$
3:  Compute $V_\rho(i) \quad \forall i \in V$
4:  $\gamma \leftarrow 1$
5:  Set a value for $M \in [4, 5]$
6:  Set a value for $\alpha \in (0.2,\ 2)$
7:  for $t \in 1, ..., T$ do
8:      Compute $conv_i^t \quad \forall i \in N$
9:      Sort $N$ by decreasing $conv_i^t$
10:     Compute $k_{max}$
11:     $k \leftarrow min(k_{max},\ k_\alpha)$
12:     Select the top $k$ customers in sorted $N$
```

2.1.4 Neighbourhood Policy: second version

The second version of *Neighbourhood Policy* (*NP_1*) starts from some considerations on *NP*: *NP* introduces customers to cells aggregation allowing the pre-computation of approximated saving indexes $\widetilde{I}_{ij}$ only once and giving a good approximation also for future customers, whose locations are unknown. However, there is some more information that is not fully exploited in *NP*:

- apart for the definition of $V_\rho(i)$, the approximated savings indexes do not enter in the definition of $conv_i^t$;
- if a pending customer is very near to the depot and his service time is small the corresponding order is quite easy to insert in a route, so the selection of these customers is not so relevant, whilst farther customers require more care;
- the one-step probability p_{ij}^t refers only to the possibility of a demand from customer j in the neighbourhood of customer i in day $t+1$, but a more precise probability can be defined taking into account the possibility that the demand of customer j is within the expiration date of order i.

These observations lead to the introduction of the second main concept in *Neighbourhood Policy*, which is the comparison of actual vs expected savings.

In *NP_1*, the definition of neighbourhood is based, as in *NP*, on cells using $V_\rho(i)$ and for each pending customer $i \in N$, a corresponding cell $i \in V$ is activated. The set of active cells at day t is still denoted by V^t, whilst the set of inactive cells is $V \setminus V^t$. For each customer $i \in N$, the set $V_\rho(i)$ is partitioned in $V_\rho(i) \cap V^t$ the set of the actual neighbours and $V_\rho(i) \setminus V^t$ the set of the possible future neighbours.
Then $\forall i \in N$, we can compute the total actual saving index, that is made by summing up all the saving indexes of the customers in the actual neighbourhood

$$\sum_{j \in V_\rho(i) \cap V^t} \widetilde{I}_{ij} \qquad i \in N \tag{2.2}$$

Similarly, we can define the total expected future savings as

$$\sum_{j \in V_\rho(i) \setminus V^t} \tilde{p}_{ij}^t \widetilde{I}_{ij} \qquad i \in N \tag{2.3}$$

In this last equation, the term $\tilde{p}_{ij}^t$ represents the probability that at least one customer j, in his corresponding cell $j \in V_\rho(i) \setminus V^t$, will require service within the last available day b_i of customer i.
To compute $\tilde{p}_{ij}^t$, we consider the uniform distribution $X \sim U([175, 215])$ from which we sample the number of new customers each day, then we use the probability p_j associated to each cell $j \in V$ in the multinomial distribution that we use to assign each new customer to his corresponding cell.
Then the expected value of X, that is the average number of new customers each day is $\overline{X} = 195$, so, if the left available time to serve customer $i \in N$ is $b_i - t$, the expected number of new customers within that period is $\overline{X}(b_i - t)$. Thus, the distribution that describes the number of future customers belonging to cell

$j \in V$ within $[t,\ b_i]$ is X_j and it is a multinomial distribution with probabilities p_h associated to each cell $h \in V$ and $\overline{X}(b_i - t)$ trials.
So, we can compute $\tilde{p}^t_{ij}$ in the following way:

$$\mathbb{P}(X_j = 0) = (1 - p_j)^{\overline{X}(b_i - t)} \quad \Rightarrow \quad \tilde{p}^t_{ij} = \mathbb{P}(X_j \geq 1) = 1 - (1 - p_j)^{\overline{X}(b_i - t)}$$

Successively, to express the difficulty to serve a customer in term of total working time associated with the order, we compute

$$d_{max} = max\{d_i\}_{i \in N}$$
$$c_{max} = max\{\tilde{c}_{0i}\}_{i \in N}$$

then we associate to each pending customer i the value

$$\beta_i = \frac{d_i + \tilde{c}_{0i}}{d_{max} + c_{max}} \qquad i \in N \tag{2.4}$$

Finally, equations (2.2), (2.3), (2.4) are combined together to express, through index $conv\_1^t_i$, the profit of selecting customer $i \in N$:

$$conv\_1^t_i = \begin{cases} \dfrac{1}{b_i - t}\left((1 - \beta_i) \times \sum\limits_{j \in V_\rho(i) \cap V^t} \widetilde{I}_{ij} - \beta_i \times \sum\limits_{j \in V_\rho(i) \setminus V^t} \tilde{p}^t_{ij}\widetilde{I}_{ij}\right) & b_i > t \\ M_1 & b_i = t \end{cases} \tag{2.5}$$

The main concepts behind this formulation are that

- orders nearer to expiration date are more urgent, so $conv\_1^t_i$ scales as the inverse of the time availability $b_i - t$;
- β_i is used to scale the contribution of actual and expected future orders: if an order is very difficult to serve $\beta_i \simeq 1$, then the expected future savings (2.3) has to be weighted more and actual savings (2.2) has less importance;
- the weighted difference between actual savings and expected future savings suggests if select or not the order: the more negative is the result of the difference the more we expect to benefit from the procrastination of the order.

It can be noticed that in (2.5) we do not consider anymore the yet postponed orders, this is done because with *NP* we managed to reach improving results with no postponed orders and the *NP_1* is implemented wishing to further improve the results, but still keeping the number of postponed orders to 0.

Then the customers' selection at day t is performed by sorting the customers by decreasing $conv\_1^t_i$, and taking the top k with

$$k = min(k_{max},\ k_{\alpha_1})$$
$$k_{\alpha_1} = |\{i \in N : conv^t_i \geq \alpha_1\}|$$

So, in the case of *NP_1* the parameters to set are ρ, M_1, α_1. Concerning ρ, probably a good threshold for the neighbourhood definition is the value found for *NP*, the value of M_1 cannot be estimated with the previous approach, the only thing that can be said about it is that $M_1 > 0$ and it has to be sufficiently large to avoid postponed orders.

To sum up, the following algorithm reports the main steps of *NP_1* implementation.

Algorithm 3 *NP_1* for customers selection

```
1:  Compute $\widetilde{I}_{ij} \quad \forall i, j \in V$
2:  Set a value for $\rho \in (0,\ 0.5]$
3:  Compute $V_\rho(i) \quad \forall i \in V$
4:  Set a value for $M_1 > 0$
5:  Set a value for $\alpha_1$
6:  for $t \in 1, ..., T$ do
7:      Compute $conv\_1_i^t \quad \forall i \in N$
8:      Sort $N$ by decreasing $conv\_1_i^t$
9:      Compute $k_{max}$
10:     $k \rightarrow min(k_{max},\ k_{\alpha_1})$
11:     Select the top $k$ customers in sorted $N$
```

2.2 Analysis of Results

Python implementation

In this section, we report the results obtained by implementing the four described policies and applying them to the simulated datasets. The reference code has been developed using *Python* on *VisualStudio Code* editor and it is entirely available at my *GitHub* repository.
Although the code has been developed using *Windows10* environment the simulations were performed using *Google Colaboratory* notepads because they offer a very useful interface with *GitHub* repositories and allow multiple simulations to be run at the same time. Since the *OR-Tools* solver does not need *GPU*, the runtime type for the simulations on *Google Colaboratory* was set to *None*.

Algorithm 4 summarizes the main steps of the simulations, that can be executed by calling the function `main.py` with the appropriate arguments:

```
python main.py file_path -p policy -d days_simulation -s solver
```

where `file_path` is the path to the file containing the distribution of customers' demands in the considered region, in the reference code it is `grid.txt`; `policy` can be chosen among values `EP`, `DP`, `NP`, `NP_1`; `days_simulation` is the number of days in the simulation, in following results `days_simulation` is equal to `100`. Finally, `solver` is the solver used to solve the instances of CVRP, in this section `solver` is set to `ortools`

All the constant values that define the data of the problem, such as the number of vehicles, the maximum duration of the routes etc. can be found and modified in file `constant.py`.

Finally the more important functions used for the policies implementation are:

`select_compatible_cells(df_distribution, depot, rho)`

that is used to create the neighbourhood $V_\rho(i)\ \forall i \in V$.
The definition of this function and a more detailed description can be found in `Functions/CostomerCompatibility.py`

`_early_policy(customer_df, this_day, num_deliveries)`
that selects customers for *CVRP* according to *EP*.

`_delayed_policy(customer_df, this_day, num_deliveries)`
that selects customers for *CVRP* according to *DP*.

`_neighbourhood_policy(customer_df, this_day, num_deliveries, compatibility, probabilities)`
that selects customers for *CVRP* according to *NP*.

`_neighbourhood_policy_1(customer_df, this_day, num_deliveries, compatibility, probabilities, compatibility_index, depot_distance)`
that selects customers for *CVRP* according to *NP_1*.

The definitions and some more detailed descriptions of the policies' functions can be found in `Functions/CostomerSelection.py`

Algorithm 4 Python Simulation

```
 1: Choose a policy $\in \{EP,\ DP,\ NP,\ NP\_1\}$
 2: Load the distribution of orders in the considered region
 3: Set the parameters' values for the policies
 4: if policy $==\ NP$ or $NP\_1$ then
 5:     Compute $\widetilde{I}_{ij} \quad \forall i,j \in V$
 6: for $t \in 1,...,T$ do
 7:     Sample from the orders distribution the new customers' demands
 8:     Update set of pending customers $N$
 9:     Apply the chosen policy
10:     Save daily data: selected customers, routes and costs
11:     Remove the selected customers from $N$
12: Compute metrics to evaluate the performance of the policy
```

Some further considerations have to be made concerning the following simulations' results.
The first 10 days in the simulations are not considered in the computation of the metrics since we start with an empty set of pending customers, but this does not correspond to the real case situation.
The metrics we use to compare the results are the average number of vehicles used each day $\bar{m}$, the average daily travel cost $\bar{c}$ and the total number of postponed orders *post*. It must be noticed that in the computation of $\bar{c}$ we do not consider the service times because those are costs that cannot be minimized.

Moreover, the choice of analysing the average daily travel cost instead of the total travel cost up to last day T is justified by the fact that, reducing $\bar{c}$, also the total travel cost is minimized and the values are smaller, so easier to compare. The metrics cited above concern the effectiveness of the policies, anyway also the execution time of the different policies can be relevant, so a comparison of the execution times has to be made.

NP results

To evaluate the *NP* we compare the obtained results with the ones of *EP* and *DP*, using different seeds for the pseudo-random generation of the daily datasets.

In the following tables, we report the values of the performance metrics $\bar{m}$, $\bar{c}$ and *post* for *EP* and *DP*.

Average travel cost	seed=57	seed=43	seed=2	seed=89	seed=1074	seed=551
EP	5580.0	5506.0	5552.0	5601.0	5507.0	5566.0
DP	5551.0	5524.0	5571.0	5603.0	5497.0	5566.0

Table 2.1: Average travel cost $\bar{c}$ for *EP* and *DP* for different seeds.

Average used vehicles	seed=57	seed=43	seed=2	seed=89	seed=1074	seed=551
EP	45.429	44.912	45.374	45.538	45.099	45.088
DP	45.319	45.066	45.462	45.659	45.077	45.077

Table 2.2: Average number of used vehicles $\bar{m}$ for *EP* and *DP* for different seeds

Postponed orders	seed=57	seed=43	seed=2	seed=89	seed=1074	seed=551
EP	0	0	0	0	0	0
DP	197	403	276	226	177	141

Table 2.3: Total number of postponed orders *post* for *EP* and *DP* for different seeds

Looking at these results, we can observe that $\bar{m}$ is quite the same for both policies and $\bar{c}$ does not seem to point out a best policy between *EP* and *DP*. Anyway, the values of *post* evidence that, in the long run, *DP* produces a lot of postponed orders, so it is a policy to be avoided for our datasets.

The first step to evaluate the performance of *NP* has been a grid search approach for values of ρ, M, α. In particular starting from the ranges $\rho \in (0, 0.5]$, $M \in [4, 5]$, $\alpha \in (0.2, 2)$, a study of $\bar{c}$, $\bar{m}$ and *post* has been performed. The values chosen for the parameters were:

- $M \in \{4,\ 4.5,\ 5\}$
- $\rho \in \{0.1,\ 0.2,\ 0.3,\ 0.4\}$
- $M \in \{0.3,\ 0.7,\ 1.1,\ 1.5,\ 1.9\}$

The values of $\bar{c}$ are reported in table 2.4. Concerning $\bar{m}$, we produced the boxplots for each seed in figure 2.2. Finally, the values of *post* are not represented since they were all equal to 0.

This first approach did not produce good results, as can be seen, the minimum values of $\bar{c}$ for each seed, the one corresponding to green cells in table 2.4, are not improving the result with respect to *EP* and *DP*. Also, the value of $\bar{m}$ does not seem to reduce when applying policy *NP*.
The only positive result about *NP* in the grid search approach is that it does not produce postponed orders.

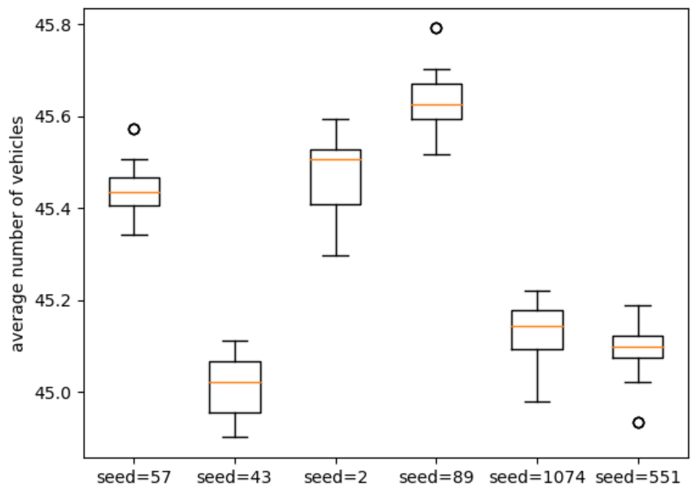
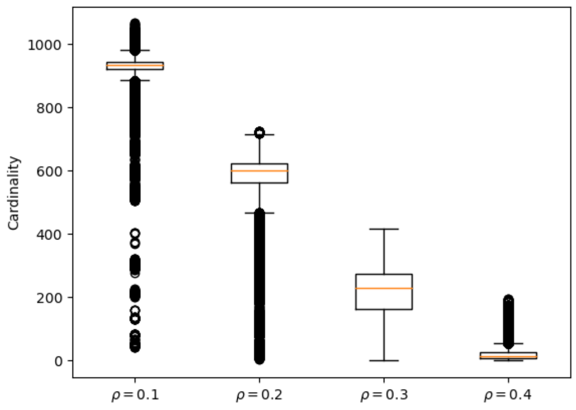


Figure 2.2: Boxplot of the average number of used vehicles using *NP* for each seed. The samples in each boxplot are produced making M, ρ and α vary in the grid search.

NP average travel cost	seed=57	seed=43	seed=2	seed=89	seed=1074	seed=551
$M = 4,\ \rho = 0.1,\ \alpha = 0.3$	5581.0	5517.0	5564.0	5598.0	5513.0	5563.0
$M = 4,\ \rho = 0.1,\ \alpha = 0.7$	5567.0	5515.0	5580.0	5584.0	5507.0	5572.0
$M = 4,\ \rho = 0.1,\ \alpha = 1.1$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 4,\ \rho = 0.1,\ \alpha = 1.5$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 4,\ \rho = 0.1,\ \alpha = 1.9$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 4,\ \rho = 0.2,\ \alpha = 0.3$	5579.0	5515.0	5561.0	5628.0	5497.0	5563.0
$M = 4,\ \rho = 0.2,\ \alpha = 0.7$	5568.0	5525.0	5579.0	5584.0	5510.0	5571.0
$M = 4,\ \rho = 0.2,\ \alpha = 1.1$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 4,\ \rho = 0.2,\ \alpha = 1.5$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 4,\ \rho = 0.2,\ \alpha = 1.9$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 4,\ \rho = 0.3,\ \alpha = 0.3$	5576.0	5519.0	5565.0	5616.0	5514.0	5567.0
$M = 4,\ \rho = 0.3,\ \alpha = 0.7$	5570.0	5517.0	5579.0	5594.0	5511.0	5574.0
$M = 4,\ \rho = 0.3,\ \alpha = 1.1$	5569.0	5513.0	5592.0	5601.0	5501.0	5586.0
$M = 4,\ \rho = 0.3,\ \alpha = 1.5$	5568.0	5512.0	5587.0	5600.0	5498.0	5588.0
$M = 4,\ \rho = 0.3,\ \alpha = 1.9$	5568.0	5512.0	5587.0	5600.0	5498.0	5588.0
$M = 4,\ \rho = 0.4,\ \alpha = 0.3$	5579.0	5519.0	5561.0	5620.0	5503.0	5563.0
$M = 4,\ \rho = 0.4,\ \alpha = 0.7$	5571.0	5508.0	5584.0	5586.0	5499.0	5572.0
$M = 4,\ \rho = 0.4,\ \alpha = 1.1$	5565.0	5511.0	5591.0	5594.0	5512.0	5568.0
$M = 4,\ \rho = 0.4,\ \alpha = 1.5$	5569.0	5519.0	5576.0	5604.0	5514.0	5570.0
$M = 4,\ \rho = 0.4,\ \alpha = 1.9$	5569.0	5519.0	5576.0	5604.0	5514.0	5570.0
$M = 4.5,\ \rho = 0.1,\ \alpha = 0.3$	5581.0	5517.0	5564.0	5598.0	5513.0	5563.0
$M = 4.5,\ \rho = 0.1,\ \alpha = 0.7$	5567.0	5515.0	5580.0	5584.0	5507.0	5572.0
$M = 4.5,\ \rho = 0.1,\ \alpha = 1.1$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 4.5,\ \rho = 0.1,\ \alpha = 1.5$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 4.5,\ \rho = 0.1,\ \alpha = 1.9$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 4.5,\ \rho = 0.2,\ \alpha = 0.3$	5579.0	5515.0	5561.0	5628.0	5497.0	5563.0
$M = 4.5,\ \rho = 0.2,\ \alpha = 0.7$	5568.0	5525.0	5579.0	5584.0	5510.0	5571.0
$M = 4.5,\ \rho = 0.2,\ \alpha = 1.1$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 4.5,\ \rho = 0.2,\ \alpha = 1.5$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 4.5,\ \rho = 0.2,\ \alpha = 1.9$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 4.5,\ \rho = 0.3,\ \alpha = 0.3$	5576.0	5519.0	5565.0	5616.0	5514.0	5567.0
$M = 4.5,\ \rho = 0.3,\ \alpha = 0.7$	5570.0	5517.0	5579.0	5594.0	5511.0	5574.0
$M = 4.5,\ \rho = 0.3,\ \alpha = 1.1$	5571.0	5513.0	5589.0	5602.0	5501.0	5586.0
$M = 4.5,\ \rho = 0.3,\ \alpha = 1.5$	5569.0	5513.0	5592.0	5601.0	5501.0	5586.0
$M = 4.5,\ \rho = 0.3,\ \alpha = 1.9$	5568.0	5512.0	5587.0	5600.0	5498.0	5588.0
$M = 4.5,\ \rho = 0.4,\ \alpha = 0.3$	5579.0	5519.0	5561.0	5620.0	5503.0	5563.0
$M = 4.5,\ \rho = 0.4,\ \alpha = 0.7$	5571.0	5508.0	5584.0	5586.0	5499.0	5572.0
$M = 4.5,\ \rho = 0.4,\ \alpha = 1.1$	5558.0	5503.0	5586.0	5596.0	5507.0	5585.0
$M = 4.5,\ \rho = 0.4,\ \alpha = 1.5$	5565.0	5511.0	5591.0	5594.0	5512.0	5568.0
$M = 4.5,\ \rho = 0.4,\ \alpha = 1.9$	5569.0	5519.0	5576.0	5604.0	5514.0	5570.0
$M = 5,\ \rho = 0.1,\ \alpha = 0.3$	5581.0	5517.0	5564.0	5598.0	5513.0	5563.0
$M = 5,\ \rho = 0.1,\ \alpha = 0.7$	5567.0	5515.0	5580.0	5584.0	5507.0	5572.0
$M = 5,\ \rho = 0.1,\ \alpha = 1.1$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 5,\ \rho = 0.1,\ \alpha = 1.5$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 5,\ \rho = 0.1,\ \alpha = 1.9$	5557.0	5508.0	5572.0	5593.0	5515.0	5580.0
$M = 5,\ \rho = 0.2,\ \alpha = 0.3$	5579.0	5515.0	5561.0	5628.0	5497.0	5563.0
$M = 5,\ \rho = 0.2,\ \alpha = 0.7$	5568.0	5525.0	5579.0	5584.0	5510.0	5571.0
$M = 5,\ \rho = 0.2,\ \alpha = 1.1$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 5,\ \rho = 0.2,\ \alpha = 1.5$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 5,\ \rho = 0.2,\ \alpha = 1.9$	5561.0	5504.0	5593.0	5598.0	5516.0	5576.0
$M = 5,\ \rho = 0.3,\ \alpha = 0.3$	5576.0	5519.0	5565.0	5616.0	5514.0	5567.0
$M = 5,\ \rho = 0.3,\ \alpha = 0.7$	5570.0	5518.0	5579.0	5594.0	5511.0	5574.0
$M = 5,\ \rho = 0.3,\ \alpha = 1.1$	5571.0	5513.0	5589.0	5602.0	5501.0	5586.0
$M = 5,\ \rho = 0.3,\ \alpha = 1.5$	5569.0	5513.0	5592.0	5601.0	5501.0	5586.0
$M = 5,\ \rho = 0.3,\ \alpha = 1.9$	5568.0	5512.0	5587.0	5600.0	5498.0	5588.0
$M = 5,\ \rho = 0.4,\ \alpha = 0.3$	5579.0	5519.0	5561.0	5620.0	5503.0	5563.0
$M = 5,\ \rho = 0.4,\ \alpha = 0.7$	5571.0	5508.0	5584.0	5586.0	5499.0	5572.0
$M = 5,\ \rho = 0.4,\ \alpha = 1.1$	5558.0	5503.0	5586.0	5596.0	5507.0	5585.0
$M = 5,\ \rho = 0.4,\ \alpha = 1.5$	5565.0	5511.0	5591.0	5594.0	5512.0	5568.0
$M = 5,\ \rho = 0.4,\ \alpha = 1.9$	5569.0	5519.0	5576.0	5604.0	5514.0	5570.0

Table 2.4: Values of $\bar{c}$ using NP for each seed, applying grid search on M, ρ and α. The green cells correspond to the minimum values for $\bar{c}$ for each seed.

Then to try and improve the results of *NP* a further analysis on values of $conv_i^t$ and ρ has been made.
Firstly, we evaluated the variability of the neighbourhood and in particular of $|V_\rho(i) \setminus V^t|$ with respect to ρ. The figure 2.3 represents the cardinality variation for each order i in the simulation for seed 57.



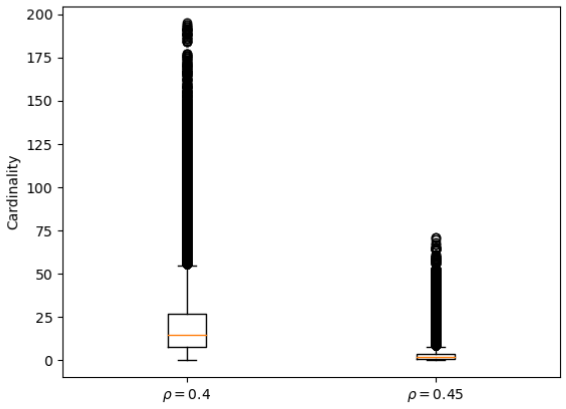


Figure 2.3: Above the boxplots for values of $|V_\rho(i) \setminus V^t|$ for different values of $\rho \in \{0.1,\ 0.2,\ 0.3,\ 0.4\}$. Below the boxplots for values of $|V_\rho(i) \setminus V^t|$ for values of $\rho \in \{0.4,\ 0.45\}$

It can be noticed that for low values of ρ the cardinality of the set is very high, this implies that the considered neighbourhood is made up of a big number of cells. Since we are considering vehicles with a limited orders capacity, $C = 5$, it does not make sense to consider a huge amount of neighbours cells in the computation of $conv_i^t$. So values of ρ like $0.1,\ 0.2,\ 0.3$ are not to be used. The values of $|V_\rho(i) \setminus V^t|$ for $\rho = 0.4$ seem to be more sensible.
A further comparison with values for $\rho = 0.45$ in figure 2.3 leads to the choice for ρ equal to 0.45 because for this value there is less variability of $|V_\rho(i) \setminus V^t|$.

The second step has been a study of the distribution of $conv_i^t$ for each customer i during the days of the simulations. The value of ρ was set to 0.45, then we collected all values of $conv_i^t$, for $b_i > t, V_\rho(i) \setminus V^t \neq \emptyset$. As can be seen in figure 2.4 we have three big bins corresponding to intervals for $conv_i^t$ which are $[0.5, 0.8]$, $[0.9, 1.1]$ and $[1.7, 2.0]$.
If we set a value of $\alpha \in [0.5, 0.8]$, we risk selecting orders that should be procrastinated, or better, to select only the top k_{max} orders obtaining a result quite similar to the one obtained by *EP*. If the value of α is taken in $[1.7, 2.0]$ then we are selecting too few customers and we do not select in a convenient day the customers whose $conv_i^t$ index never reaches such a high value unless for $b_i = t$. So, a good smaller interval for the search of α value is $[0.9,\ 1.1]$. Further analysis of the distribution in $[0.99,\ 1.0]$ suggests to look for α value very near to 1.0.

Following these considerations, we obtain also some indication for the value of M. Since we are looking for a value of $\alpha < 1.0$, a good value for M seems to be 5, because in this case $\frac{M}{b_i - t} > 2$ only for $b_i - t \in \{1, 2\}$, but if we compare orders with $V_\rho(i) \setminus V^t \neq \emptyset$ and $conv_i^t < 1.0$ with orders corresponding to $V_\rho(i) \setminus V^t = \emptyset$, the lasts will have the priority on the first ones, for any $b_i - t \in \{1, 2, 3, 4, 5\}$.

Summing up, we have reduced the search for the parameters in the following way:

- $\rho \in (0.2, 0.5) \longrightarrow \rho = 0.45$
- $M \in [4, 5] \longrightarrow M = 5$
- $\alpha \in (0.2, 2) \longrightarrow \alpha \in (0.99, 1.0)$

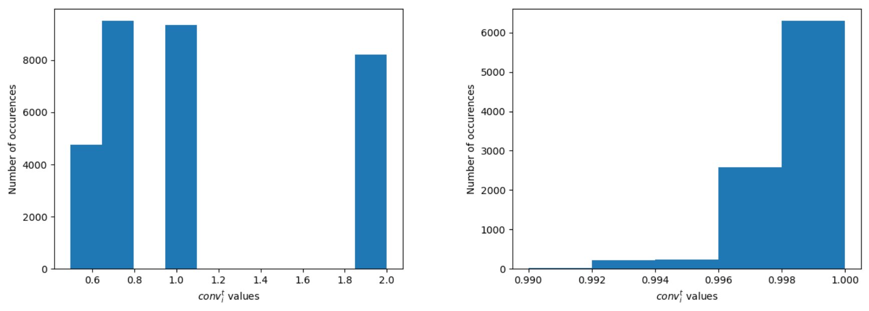


Figure 2.4: Histograms representing the number of occurrences of $conv_i^t$ values. The plot on the left represents the overall distribution of $conv_i^t$, we can notice three well-separated bins, with the same order of magnitude of occurrences. The plot on the right is a histogram based on values of the central bin of the left plot, corresponding to $conv_i^t \in [0.99,\ 1.0]$

Applying the obtained results to a gradually finer grid search on values for α we reach the final step of the tuning looking at the results of *NP* for $\alpha \in [0.999, 1.0]$ with step-size equal to 0.0001.
The figures below show $\bar{c}$, $\bar{m}$ and the total travel cost of each simulation, at the

varying of α. The obtained results are compared to the benchmarks represented by policies *EP* and *DP*, trying to find a value of α that minimizes all the considered metrics. For a less dataset-dependent choice of α, six seeds for dataset generation have been considered: $seed \in \{57,\ 43,\ 2,\ 89, 1074,\ 551\}$

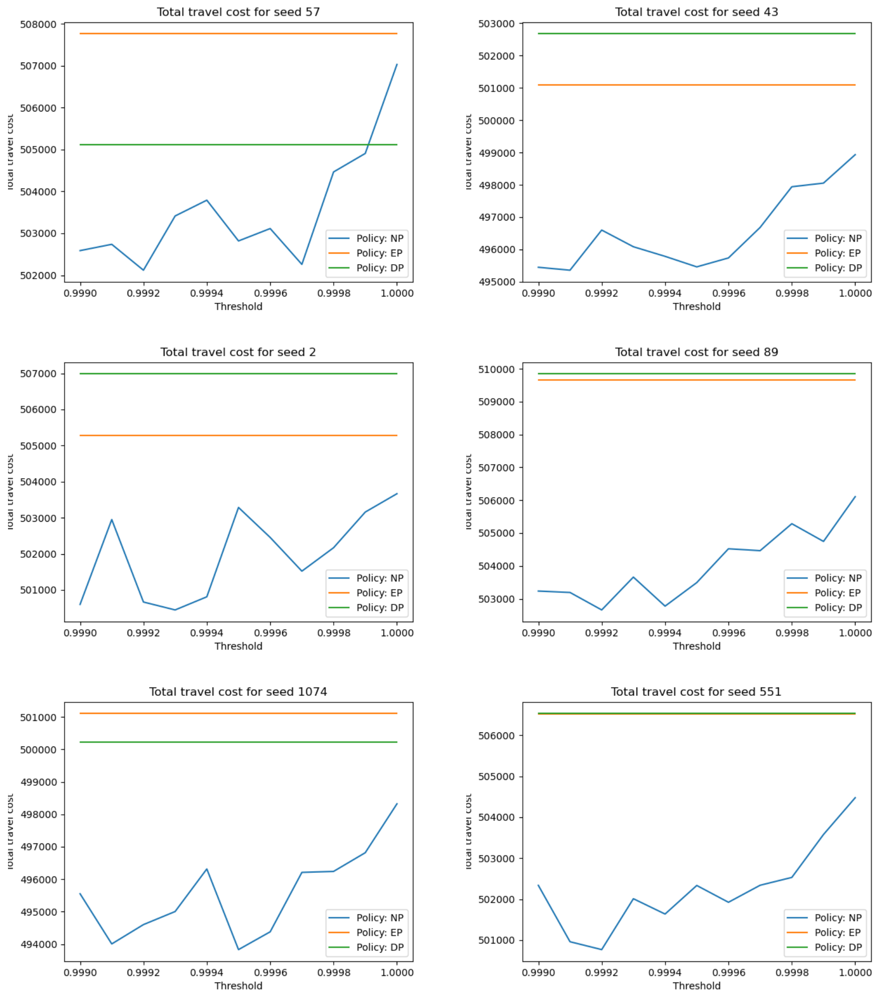


Figure 2.5: Total travel costs for different seeds applying *EP*, *DP* and *NP* with $\alpha \in [0.999, 1.0]$ with step-size equal to 0.0001, $M = 5$, $\rho = 0.45$, $\gamma = 1$. Comparing the six plots, lot of variability can be noticed in the total travel cost, anyway except for $seed = 57$ and $\alpha = 1.0$ all the costs are below the benchmarks represented by policies *EP* and *DP*. In three cases over six, $seed = 57,\ 89,\ 551$, $\alpha = 0.9992$ corresponds to the minima, for $seed = 2$ it leads to a slightly suboptimal result. So $\alpha = 0.9992$ seems a good threshold.

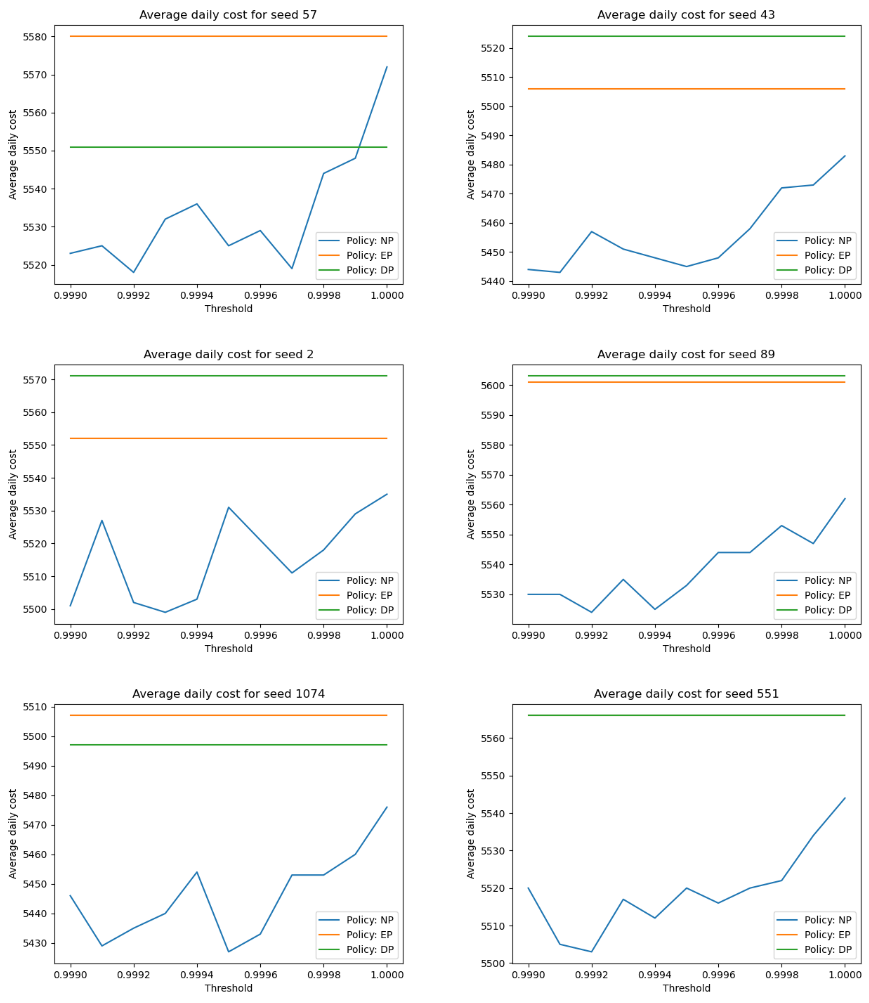


Figure 2.6: Average daily costs $\bar{c}$ for different seeds applying *EP*, *DP* and *NP* with $\alpha \in [0.999, 1.0]$ with step-size equal to 0.0001, $M = 5$, $\rho = 0.45$, $\gamma = 1$. Comparing the six plots, we can observe that the trends in the costs $\bar{c}$, for different values of α, are very similar to the ones in figure 2.5. In fact, also for $\bar{c}$ the combination $seed = 57$ and $\alpha = 1.0$ corresponds to an average cost higher than the benchmark of *DP*, all other combinations lead to costs below the benchmarks of *EP* and *DP*. In three cases over six, $seed = 57,\ 89,\ 551$, $\alpha = 0.9992$ corresponds to the minima, for $seed = 2$ it leads to a slightly suboptimal result. So, similarly to figure 2.5, $\alpha = 0.9992$ seems a good threshold.

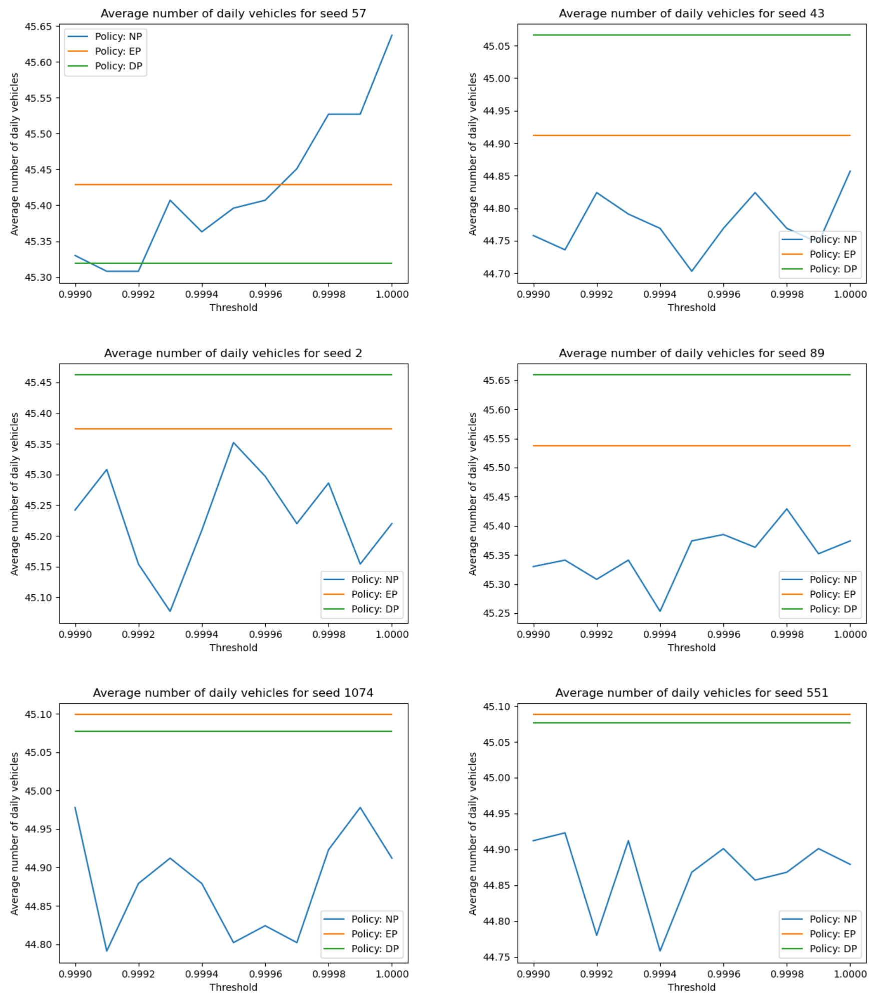


Figure 2.7: Average number of daily vehicles $\bar{m}$ for different seeds applying *EP*, *DP* and *NP* with $\alpha \in [0.999, 1.0]$ with step-size equal to 0.0001, $M = 5$, $\rho = 0.45$, $\gamma = 1$. In this case, the variation of $\bar{m}$ is in a very small interval, differences in the number of vehicles range in $[0.3,\ 0.4]$, so these results cannot be really relevant, since *NP* does not save even a whole vehicle. Anyway, looking at the plot of $seed = 57$, a suggested range for α is $[0.9991,\ 0.9992]$ since it is the only interval in which $\bar{c}$ is below the benchmarks of policies *EP* and *DP*. This result is in agreement with the results reported in figures 2.5, 2.6 .

By a majority vote, looking on best values of $\bar{c}$, $\bar{m}$ and knowing that *NP* has $post = 0$ for each value of α, the best configuration for *NP* is:

- $\rho = 0.45$
- $M = 5$
- $\gamma = 1$
- $\alpha = 0.9992$

NP_1 results

The results on policy *NP* encourage the implementation of policy *NP_1*. Moreover, the tuning approach for *NP* can be taken as a guideline for the tuning of *NP_1*. In this last policy the parameters to tune are ρ, M_1 and α_1.
In this case, the value of M_1 is less complicated to set with respect to the case of *NP*, because it is sufficient that $M_1 > conv\_1_i^t \ \forall \ b_i > t$. The choice of ρ can be made through an analysis of the cardinalities $|V_\rho(i) \cap V^t|$ and $|V_\rho(i) \setminus V^t|$ which determine the neighbourhood in the definition of $conv\_1_i^t$ in equation 2.5.
Looking at figure 2.8 we can notice that $\rho = 0.4$ produces a very big neighbourhood $V_\rho(i)$, that has a cardinality up to 200. A smaller and more suitable neighbourhood is the one defined by $\rho = 0.45$.

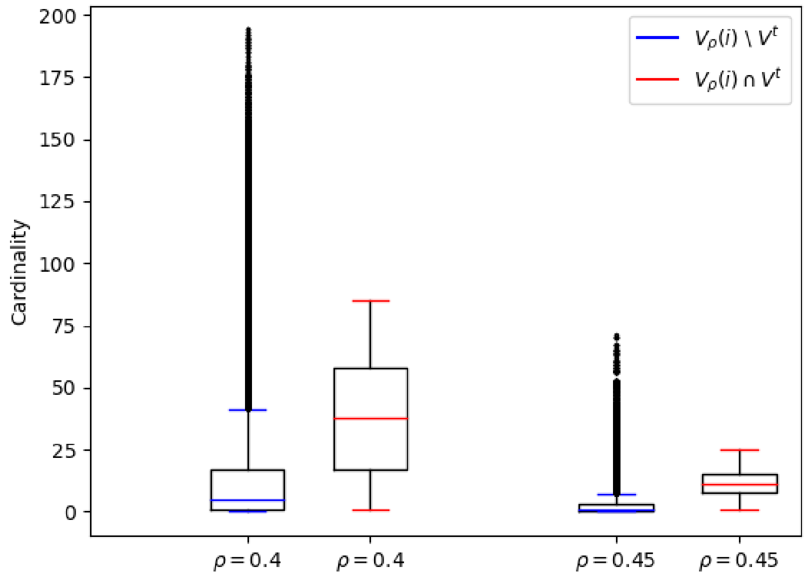
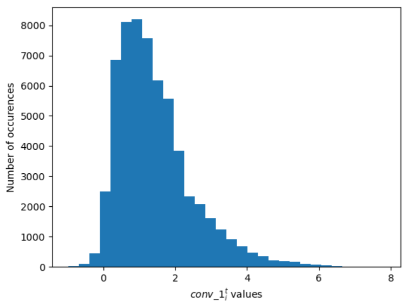


Figure 2.8: Boxplot for cardinalities $|V_\rho(i) \cap V^t|$ and $|V_\rho(i) \setminus V^t|$ for $\rho \in \{0.4,\ 0.45\}$.

Concerning the value of M_1, the analysis of values $conv\_1_i^t \ \forall \ b_i > t$ produces the histograms reported in the figure 2.9. So a $M_1 > 7$ is a value sufficiently large such that all orders with $b_i = t$ are selected, we decided to set $M_1 = 8$.
The search for an adequate value of α_1 is more complicated, in this case, we have not well-separated bins that clearly identify very low or very high values for $conv\_1_i^t$, moreover, the threshold α_1 influences a lot the indexes distribution, as can be noticed by comparing the two plot in figure 2.9.



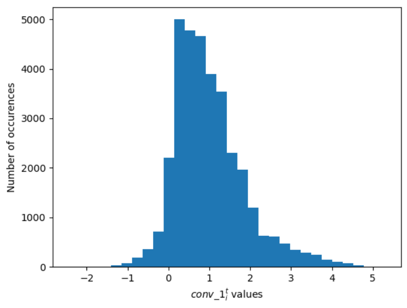


Figure 2.9: Histograms representing the number of occurrences of $conv\_1_i^t \; \forall \; b_1 > t$ values. On the left the indexes distribution when orders are selected with a threshold $\alpha_1 = 8$, on the right the distribution corresponding to threshold $\alpha_1 = 0.4$

So the tuning of parameter α_1 has been made through a gradually finer grid search:

- $\alpha_1 \in [-1.1, \; 2.3]$ with step-size equal to 0.2
- $\alpha_1 \in [-0.5, \; -0.3]$ with step-size equal to 0.02

The following figures summarize the tuning results on policy *NP_1*. We plot the values of the total travel cost up to the 100^{th} day of simulation, the average daily cost $\bar{c}$ and the average number of used vehicles $\bar{m}$ for different values of α_1. The results of *NP_1* are compared to the results of *EP* and *DP*, for all the three considered costs the best value of α_1 corresponds to the minima. To select a value of α_1 not depending on the specific values of a dataset, different seeds for the dataset generation has been used.

It can be noticed that *NP_1* further improves the results of *NP*, not only for the costs $\bar{c}$ and consequently for the total travel cost, but it also reduces significantly the number of used vehicles. Moreover, these results have been achieved without producing postponed orders ($post = 0$).
Finally, the chosen value for α_1 has been -0.4 since generally a good range for this parameter is $[-0.5, -0.3]$, but for α_1 values ranging in this interval, there are some oscillations in the costs that make difficult to set a universally good value for α_1. The value $\alpha_1 = -0.4$ is associated with the minimum number of vehicles in two cases over six, even if the minima of the travel costs correspond to other values in $[-0.5, -0.3]$. Anyway, the costs associated with $\alpha_1 = -0.4$ are only slightly suboptimal with respect to the global minima reached by costs functions in the simulated scenarios.

To sum up, the best configuration for *NP_1* found so far is:

- $\rho = 0.45$
- $M_1 = 8$
- $\alpha_1 = -0.4$

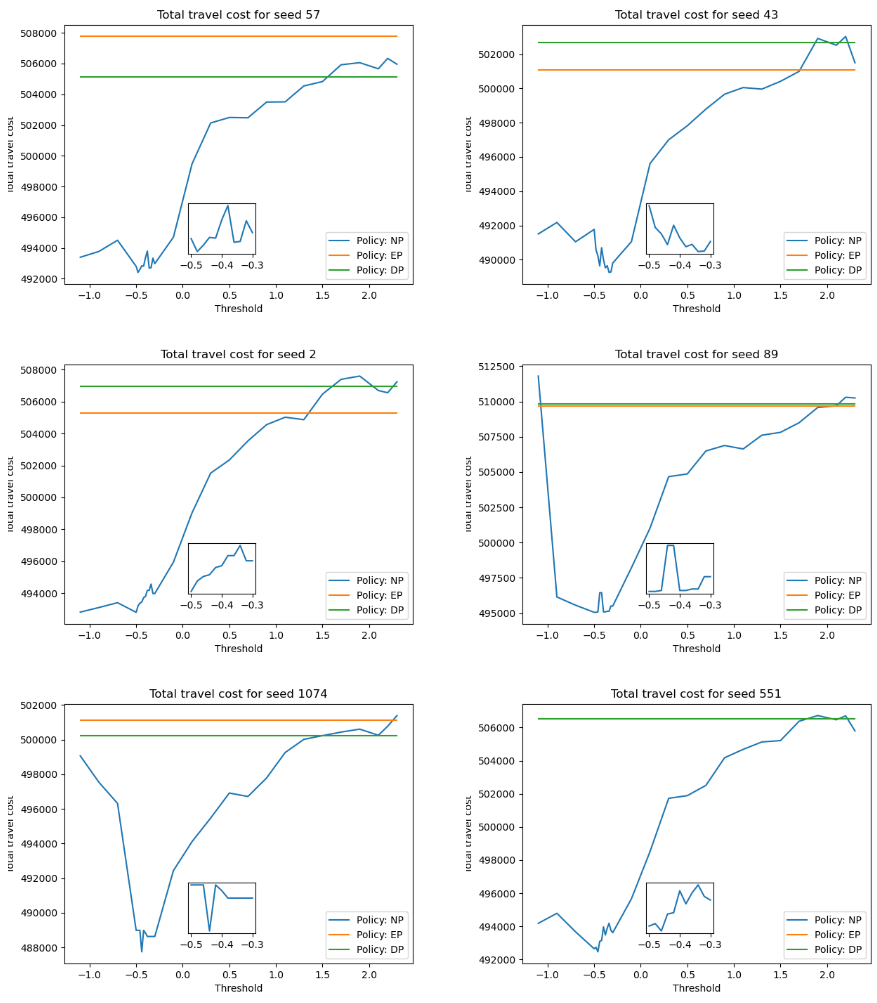


Figure 2.10: Total travel costs for different seeds applying *EP*, *DP* and *NP_1* with $\alpha_1 \in [-1.1, 2.3]$, $M_1 = 8$, $\rho = 0.45$. *NP_1* reduces the costs if α_1 threshold is set properly. In these plots, a good range for this parameter is the one highlighted by the zooms, which is $\alpha_1 \in [-0.5, -0.3]$. Concerning the minima of the cost functions, most of them are in the range $[-0.5, -0.46]$.

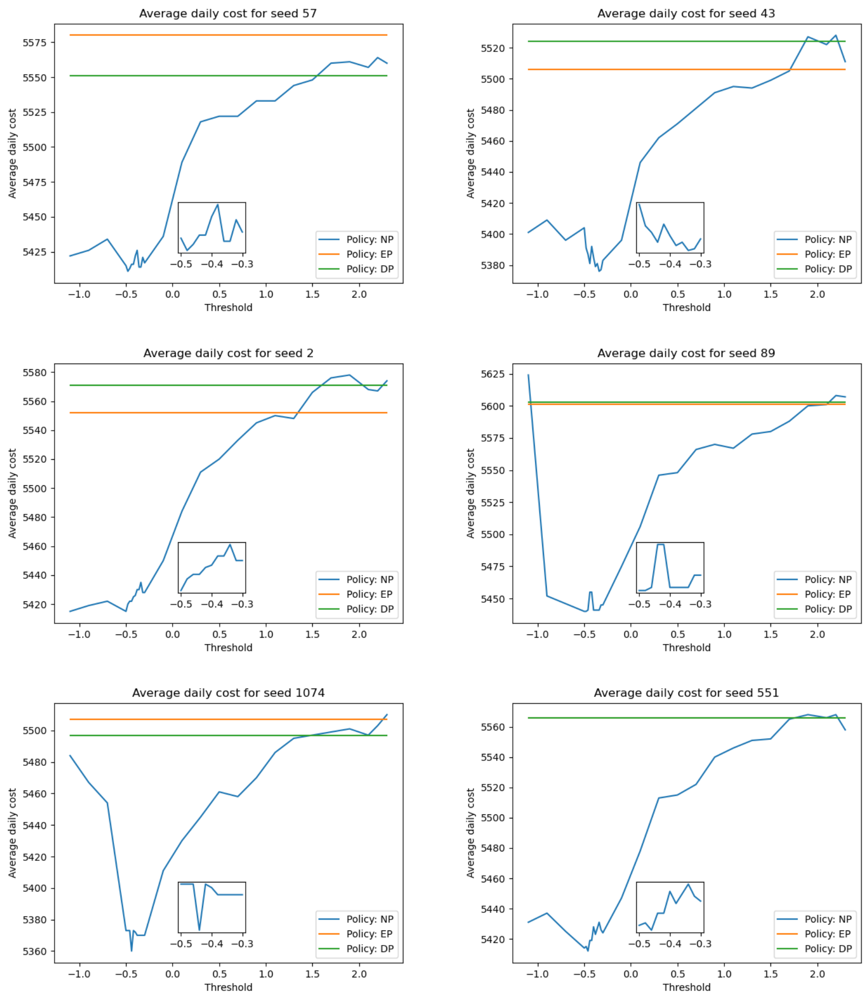


Figure 2.11: Average daily costs $\bar{c}$ for different seeds applying *EP*, *DP* and *NP_1* with $\alpha_1 \in [-1.1,\ 2.3]$, $M_1 = 8$, $\rho = 0.45$. Similarly to figure 2.10, *NP_1* reduces the average costs, especially for α_1 threshold $\in [-0.5,\ -0.3]$, this interval is the one zoomed in each figure. It can also be noticed that the trends of functions $\bar{c}$ are really similar to the ones of the total costs, that justifies the choice of considering only one of these values as a performance metric.

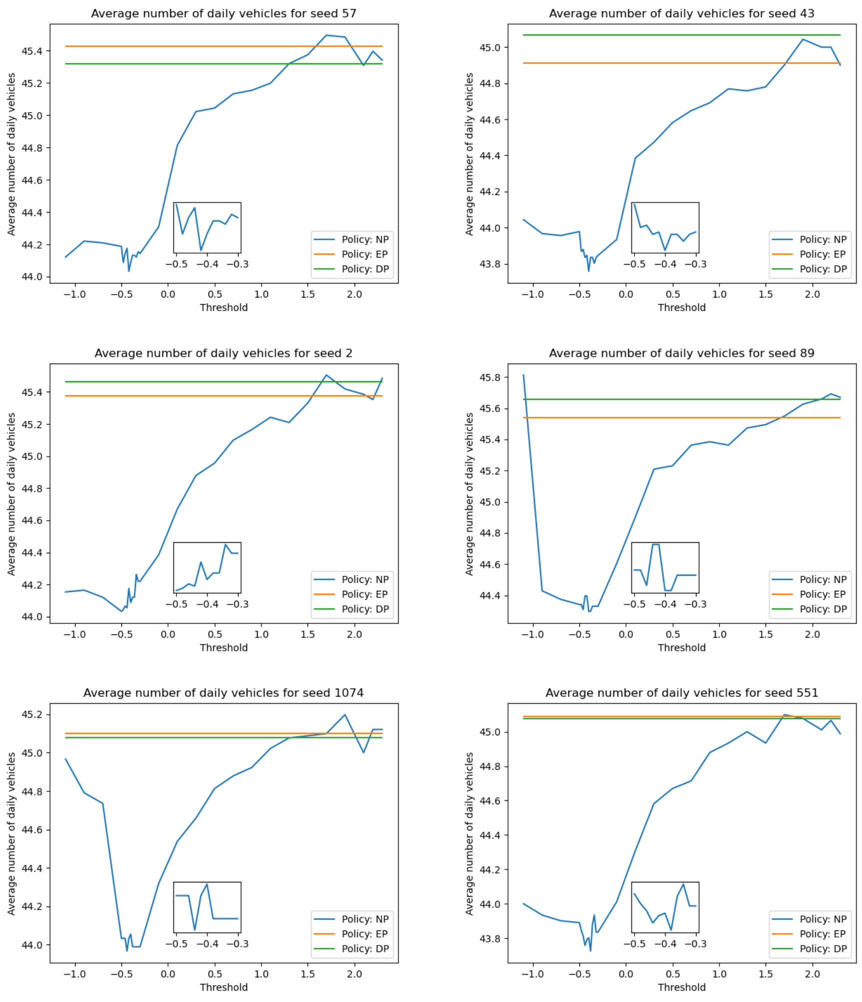


Figure 2.12: Average number of daily vehicles $\bar{m}$ for different seeds applying *EP*, *DP* and *NP_1* with $\alpha_1 \in [-1.1,\ 2.3]$, $M_1 = 8$, $\rho = 0.45$. These plots evidence that good tuning on *NP_1* can lead to a significant reduction of the number of used vehicles. For values of $\alpha_1 \in [-0.5,\ -0.3]$ we get a reduction of more than one vehicle in the average daily used ones, this can be a relevant reduction in the fixed costs, if we consider the total fixed costs on the long run. Finally, in the suggested range for the threshold α_1, the value -0.4 is associated with minima in two cases over six.

Final Comparison

In this final subsection, we compare the results of the four policies *EP*, *DP*, *NP* and *NP_1*, considering the total travel costs, the average daily costs, the average daily number of used vehicles and the execution times of the simulations. The first three metrics will evaluate the performances of the policies, whilst the execution time is a measure of the effort to compute indexes and sort customers for the selection.

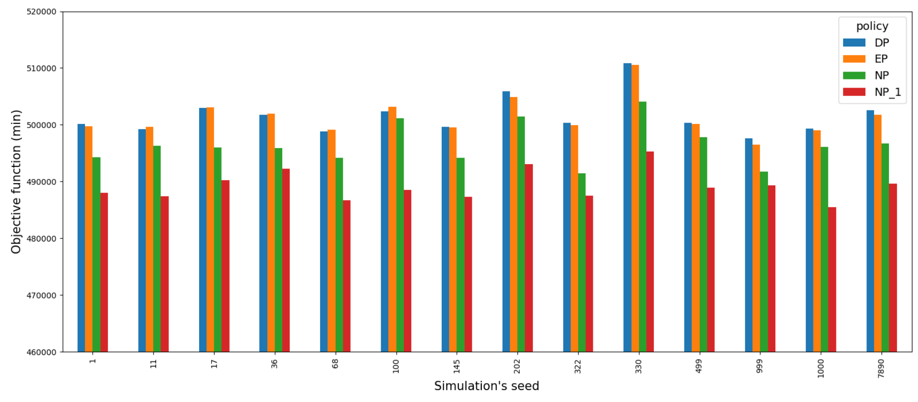


Figure 2.13: Total travel costs for datasets generated with different seeds.

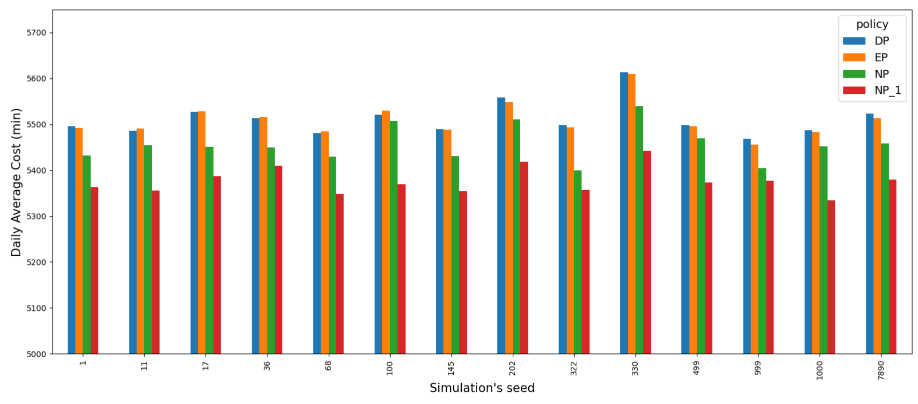


Figure 2.14: Average daily travel costs for datasets generated with different seeds.

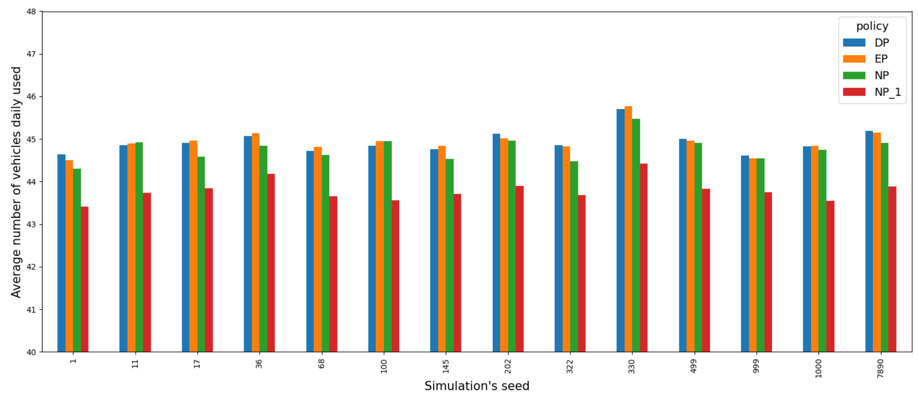


Figure 2.15: Average number of daily used vehicles for datasets generated with different seeds.

Looking at figures 2.13 and 2.14, it can be noticed that for all instances *NP* and *NP_1* introduce a relevant reduction in the costs with respect to *EP* and *DP*. Generally, *NP_1* performs better than *NP*.
Moreover, in figure 2.15 it can be observed that, for all instances, *NP_1* significantly reduces the number used vehicles, whilst *NP* uses quite the same vehicles than *EP* and *DP*.

To quantify the improvements of the neighbourhood policies we define the relative percentage improvement as

$$\%impr_* = \frac{min(cost_{EP}, cost_{DP}) - cost_*}{cost_*} \times 100 \qquad\qquad * = NP,\ NP\_1$$

The results of this formula, applied to the costs obtained by simulations are summarized in tables 2.5 and 2.6.

Average travel cost	min(*EP*,*DP*)	*NP* improvement (%)	*NP_1* improvement (%)
seed=145	5489.0	1.07	2.50
seed=68	5481.0	0.94	2.49
seed=999	5456.0	0.96	1.47
seed=202	5549.0	0.69	2.42
seed=7890	5514.0	1.03	2.49
seed=11	5486.0	0.59	2.43
seed=36	5514.0	1.19	1.94
seed=322	5494.0	1.74	2.56
seed=1	5492.0	1.10	2.41
seed=17	5527.0	1.39	2.60
seed=100	5521.0	0.25	2.83
seed=330	5610.0	1.26	3.09
seed=499	5496.0	0.48	2.29
seed=1000	5484.0	0.59	2.79

Table 2.5: Relative percentage improvements of policies *NP* and *NP_1* with respect to minima average daily costs of benchmark policies *EP* and *DP*. *NP_1* is associated with the most relevant improvements.

Average daily vehicles	min(*EP*,*DP*)	*NP* improvement (%)	*NP_1* improvement (%)
seed=145	44.758	0.52	2.39
seed=68	44.714	0.20	2.44
seed=999	44.549	0.03	1.83
seed=202	45.022	0.12	2.55
seed=7890	45.153	0.56	2.90
seed=11	44.857	-0.14	2.56
seed=36	45.066	0.49	2.01
seed=322	44.824	0.79	2.62
seed=1	44.505	0.47	2.50
seed=17	44.912	0.74	2.43
seed=100	44.835	-0.25	2.92
seed=330	45.703	0.51	2.87
seed=499	44.956	0.10	2.58
seed=1000	44.824	0.17	2.93

Table 2.6: Relative percentage improvements of policies *NP* and *NP_1* with respect to minima average number of vehicles of benchmark policies *EP* and *DP*. *NP_1* is associated with the most relevant improvements.

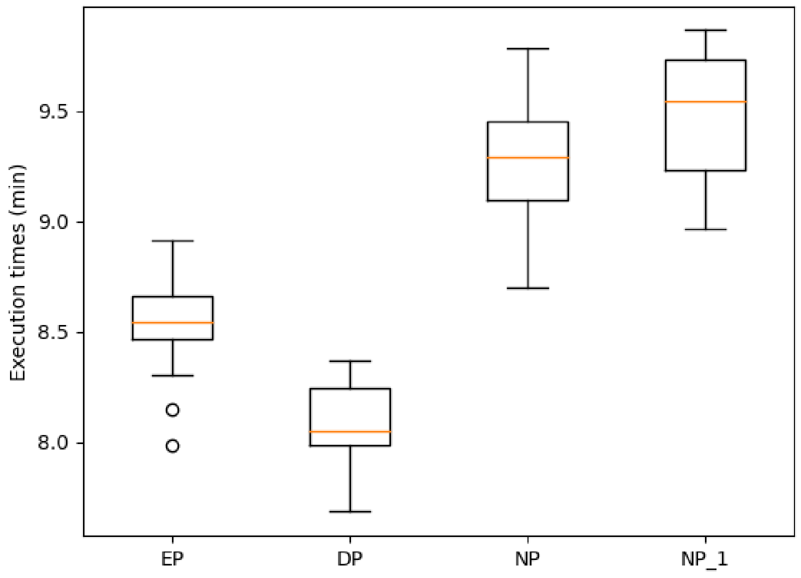


Figure 2.16: Boxplot of execution times of simulations for the policies *EP*, *DP*, *NP*, *NP_1*.

Finally, some observations about execution times have to be made: *NP* and *NP_1* require more computations than *EP* and *DP* and this is reflected in higher execution time. Anyway, the total execution time for a 100 days simulation is of about 10 minutes, which corresponds to nearly 6 seconds a day, including dataset generation, customers' selection and *CVRP* solution by means of *OR-*

Tools solver.
Looking at boxplots in figure 2.16, we can observe that the gap in the execution times introduced by *NP* and *NP_1* is of nearly 1 minute per simulation that is nearly 0.6 seconds per day in simulation. However, this slight increment is justified by the much more relevant reduction of the costs.

Chapter 3

Solvers for CVRP

In this chapter, we discuss two solvers to solve the daily *CVRPs* in the simulation.
Nowadays, there are many solvers that could be used to solve this kind of problems, and most of them are very keen on the constraints' definition. Anyway, solvers that implement exact algorithms, like *Gurobi* or *CPLEX*, require a lot of execution time for large instances and solutions obtained with a very short time limit risk to be really suboptimal.
Other software for *CVRP* based on meta-heuristics, like *VRPH*, or on constraint optimization, such as *OR-Tools* are much faster and produce very convenient results.
In the first part of this thesis, *OR-Tools* solver has been used, because it has an available library for *Python* and allows a handy definition of constraints, whilst *VRPH* is written in *C++* and the integration of the solver is possible only through an executable file that takes as input .txt files; moreover the only capacity constraint available in *VRPH* is the one concerning the number of customers visited by each vehicle.
In this second part of the thesis, we introduce a new solver, based on a meta-heuristic approach, that improves the convenience of the routes with respect to the ones found by *OR-Tools*. So, the main analysis of this chapter concern:

- *Google OR-Tools* solver: it is a solver based on constraint optimization, developed by *Google API* to solve various classes of problems: assignment, bin packing, network flows, scheduling and routing problems. This solver is available for *C++*, *Java*, *Python* and *DotNet* programming languages, so it is very easy to integrate inside the code. This kind of solver takes nearly 6 seconds to sub-optimally solve the instances of *CVRP* produced by the customers' selection.

- *CW-TS* solver: it is a solver developed during this thesis, which aims to improve the results of *OR-Tools*, but maintaining the definition of all the constraints. This solver has been implemented using *Python* to be easily included in the code developed in the first part of the thesis. Anyway, the Object-Oriented structure of the code allows an easy transcription to compiled and faster languages, such as *C++* and *Java*. This solver is composed of two main steps:

- *Clark-Wright* and *SmallRoutesElimination* algorithms: greedy initialization of the *CVRP* solution. This first solution is dominated by the *OR-Tools* one and runs in less than 1 second.
- *TabuSearch* meta-heuristic: meta-heuristic that produces at each step a new feasible solution of the *CVRP*, by means of *Swap* and *Insertion* functions combined with a *Local Search*. The exploration of the solution space is guided by a *Tabu Search* approach that highlights tabu moves and avoids local minima. This second phase of the algorithm is the one that produces improving routes and it is designed to run for 45 seconds.

3.1 *Google OR-Tools* solver

OR-Tools is open-source software for combinatorial optimization, its library is a set of operations research tools developed in *C++* by *Google*. It can be used to find the optimal, or suboptimal, solutions to an optimization problem, using state-of-the-art algorithms to narrow down the exploration of the solution space. The performances of *OR-Tools* won it four gold medals in the 2019 *MiniZinc Challenge*, the international constraint programming competition.

Concerning the routing optimization, *OR-Tools* can be used to solve different kinds of problems:

- *TSP*: the most basic routing problem.
- *CVRP*: *VRP* with capacity constraints.
- *TWVRP*: *VRP* with time windows that specifies feasible time intervals to visit the customers
- *VRP with dropped visit*: *VRP* in which vehicles could visit only a subset of all customers, but take a penalty for each customer that is not visited (dropped customer).

For all these problems, an exhaustive search on the solution space would lead to the optimal solution. Anyway, the amount of execution time to solve them grows exponentially with the size of the problem, determined by the number of customers. So, an exact approach is computationally intractable for all but small sets of locations. Considering larger problems, optimization techniques are needed to investigate the solution space in a smart way and find optimal or sub-optimal solutions.
The approach used by *OR-Tools* is a combination of constraint programming and meta-heuristics, based on *Local Search* and *Large Neighbourhood Search*.

3.1.1 Constraint Programming

Constraint Programming, in brief CP, is a field of operations research that aims to solve optimization problems focusing on the constraints rather than the objective functions.
CP has been successfully applied to different kinds of optimization problems, also with heterogeneous constraints. In fact, one of the main strengths of CP is the possibility to deal with any kind of constraints, even the ones that include symbolic variables and meta-constraints i.e. constraints on constraints.
Another peculiarity of CP is the ease of modelling a problem: since CP exploits locally each constraint, the insertion of constraints, even during the search, can be easily dealt with.
These aspects result in a very important advantage in CP: CP can be applied to very different kind of models, maintaining quite the same search strategy.

The problems to which CP is applied are called Constraint Satisfaction Problems (CSP). They are mathematical models with constraints and the aim is to find a feasible solution. That is done by assigning values to the decision variables of the model such that every constraint is met.

Once defined the model, some algorithmic techniques are applied in order to solve the problem: these techniques aim to find a smart way to visit the search space reducing the number of visited solutions and investigating only the interesting portions of the search space as quickly as possible.
The main techniques used in CP are *chronological backtracking*, *constraint propagation*, *Local Search* and *branching heuristics*.

- *Backtracking* search is a general algorithm to solve CSP that systematically generates candidates to the solutions, by assigning feasible values to the variables. It tries to extend a partial solution to a complete one, but if it finds infeasibility during the process, it backtracks: it abandons the candidate and returns to the last valid assignment.

- *Constraint propagation* reduces the domains of the variables, strengthening or creating new constraints. It aims to make the problem easier to solve and can be used also to check infeasibility in some portions of the search space. If the application of constraint propagation does not lead to a solution, a feasible value is assigned to a variable.

- *Local Search* is an incomplete method to improve the solution of a problem. It proceeds iteratively assigning values to variables until all constraints are satisfied: more precisely, only one value of a variable is modified in each iteration, this leads to a solution that is close in the search space to the one of the previous iteration and justifies the name local.

- *Branching heuristics* are heuristics based on Branch&Bound method, since they are not exact methods they not necessarily lead to optimal solution, but are designed to avoid wasting execution time in the exploration of worsening solutions. Some of the most famous branching heuristics combine pruning (to propagate constraints and reduce the variables domain), creation of cuts (new constraints to further reduce the search space) and beams based approaches (to reduce the number of investigated branches in the search tree).

Usually, these techniques are combined together and exploit the dynamic programming principle. Dynamic programming is both a mathematical optimization method and a computer programming method. It consists of simplifying a complicated problem by splitting it up into easier sub-problems recursively and then solve the easiest problems. It must be noticed that non all decision problems can be solved with this approach, but in general decisions that are taken in several stages allow a recursive formulation.

3.2 ***CW-TS*** solver

The *OR-Tools* solver allows us to obtain good results in a very short time, anyway, the user has little control over the algorithm of the solver and it has to use it as a black-box solver, given the data of the problem. So, to try and further improve the results, a meta-heuristic to solve *CVRP* has been developed.
This meta-heuristic, that has been named *CW-TS* algorithm, is the combination of three famous algorithmic techniques: *Clark-Wright* algorithm, *Tabu Search*

heuristic and *Local Search.*

The first step in *CW-TS* algorithm is the initialization of a feasible solution, that is to find non-overlapping routes that visit all customers, satisfying all capacity constraints (on the number of customers, on the maximum load and on the duration of the route) and using a feasible number of vehicles. To initialize routes *Clark-Wright* algorithm has been combined with *SmallRoutesElimination* algotithm.
Clark-Wright algorithm starts with the computation of the savings matrix, that is, given the set of selected customers S and the depot location 0, the entry at row i and column j of the savings matrix SM is

$$SM_{ij} = \begin{cases} c_{0i} + c_{0j} - c_{ij} & \forall\ i \neq j \in S \\ 0 & \forall\ i = j \in S \end{cases}$$

where $c_{ij}\ \forall i, j \in S \cup \{0\}$ are the travel costs between the locations.
The entries of SM, analogously to savings indexes I_{ij}, describe the convenience of visiting customers i and j with the same route, the higher the savings the more convenient is their assignment to the same route.
Then each customer is assigned to a route, so at the beginning, the number of vehicles is equal to the number of customers: each vehicle starts from the depot, visits exactly one customer and returns to the depot. Each route is correctly initialised, by updating the current load, the number of visited locations, the duration and the path of the route. After that, the entries in the upper triangular matrix of SM, except the diagonal elements are sorted in decreasing order: the row-column pairs indexes (i, j) correspond to customers in S. Given the sorted pairs of customers, we consider the corresponding routes and try to merge them. If the merge leads to a feasible route, then the set of current routes is updated: the pair of old merged routes are eliminated, the new merged route is inserted with the updated path, load and duration, and the number of used vehicles is decremented by one.
This approach is applied to all sorted pairs of customers, so hopefully, in the end, we obtain fewer routes that satisfy the capacity constraints.
Given these routes, we apply *SmallRoutesElimination* algorithm: the routes obtained by *Clark-Wright* algorithm can include small routes, i.e. routes that visit only 1 or 2 customers. These routes can probably be eliminated or better, the number of small routes, and consequently, the number of used vehicles, could be reduced. So, *SmallRoutesElimination* acts by considering each customer in the small routes and trying to insert him in another route. The insertion is done by looking for the best feasible insertion for the customer, which is the insertion that produces the lowest additional travel cost, still preserving the feasibility of the chosen route for the insertion. If a feasible insertion is found, the two routes involved in the insertion and the set of small routes are updated.
At the end of this step, if the number of used vehicles is lower than the number of available vehicles, we obtain a feasible initial solution. Otherwise, the last customer in S is removed and the initialization of routes is repeated.

The second step of *CW-TS* algorithm concerns the *Tabu Search* and *Local Search* techniques.
Given the initial feasible solution, we apply an iterative procedure that aims

to reduce the travel costs of the routes. The number of iterations is implicitly controlled by setting a time limit for the execution of the algorithm.
The *Tabu Search* approach is used to avoid getting stuck in local minima: given the current solution, we generate a solution in its neighbourhood, that is a solution that is a slight modification of the current one. In our case, a neighbour solution is obtained by slightly modifying at most 4 routes. Then, the cost of the neighbour solution is evaluated and we decide if accept or not this solution as a new current one. Anyway, to decide whether to accept or not a solution we cannot consider only the improvement on the travel costs, because if we are in the neighbourhood of a local minimum this would lead to a solution that gets trapped in the local minimum. The main idea is that we have to accept some worsening solution to avoid local minima and go towards global optima.
Tabu Search expresses this concept introducing the tabu list and the aspiration criterion:

- The tabu list is a list that contains tabu moves, that are variations of the current solution that lead to a yet visited solution. In our case, before generating the neighbour solution, we save the old routes' paths of the routes involved in the transformation. Then, if the neighbour solution is accepted, we insert in the tabu list the reverse moves for the neighbourhood, which are the old routes' paths. In this way, each time we generate a neighbour, we can easily check if the new routes correspond to a yet visited solution, this corresponds to a tabu violation. Another key element for the tabu list is its length *tabu_length*, in fact, we do not memorize in the tabu list all the reverse moves, but we set a fixed maximum length for the tabu list and add the reverse moves till the length of the tabu list reaches the maximum length. Once reached the maximum dimension, the tabu list is updated in FIFO mode, allowing the memorization of the most recent tabu moves and losing memory of very old tabu moves.

- The aspiration criterion is used to determine the acceptance of a solution. In fact, given the neighbour solution there are three aspects that characterize it: the difference $diff$ in travel costs between the current solution and the neighbour one, the difference $diff\_best$ in travel costs between the best solution found so far and the neighbour solution, the reduction of used vehicles associated with the neighbour solution and the violation of the tabu in the neighbour generation. If the $diff > 0$ and the neighbour solution does not violate the tabu, the solution is always accepted, if $diff < 0$, the neighbour solution does not violate the tabu and we have not accepted a new solution within the last *gap_worse* iterations, we accept the worsening solution. Finally, the neighbour solution is accepted, even if it violates the tabu, if it reduces the number of used vehicles or it is associated with the minimum travel cost found so far, $diff\_best > 0$, that is the aspiration criterion.

Another critical point in this second step of *CW-TS* algorithm concerns the neighbours generation, in fact, a compromise between the simplicity and the improvement of the neighbourhood has to be reached: if we use too simple and random algorithms to produce a neighbour solution, we risk needing too many iterations to reach an improving solution, if we use too complicated and optimum-seeking algorithms we risk to waste too much execution time in the

neighbour generation and to point towards local minima.
After various attempts, the best configuration to generate a neighbour has been found as a combination of *Swap* and *Insertion* algorithms.
The *Swap* consists of selecting two random routes in the current solution, then one random customer is selected on each of the two routes. These customers are intended to be swapped, that is, the customer selected on the first route is assigned to the second route and vice-versa, the order of the other customers in the routes is preserved. Every time a swap is tried, the feasibility of the new routes is checked, if the swap is not feasible, other two random routes with corresponding customers are selected, until a feasible swap is found.

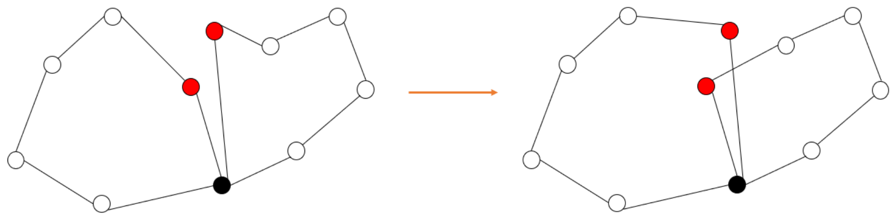


Figure 3.1: Example of routes generated by *Swap*. The routes on the left are the two randomly selected ones, the customers selected for the swap are the red ones and the depot is represented by the black circle. The routes obtained with the swap are the ones on the right.

The insertion phase takes place after having found a feasible swap. It consists of operating on two random routes, if possible the first route is selected among small routes. Then a random customer is selected on the first route, given this customer, we try to insert him in the best position in the second route. If the insertion leads to a feasible route then both routes are updated and contribute to the neighbour solution definition, otherwise, the perturbation of the current solution is only given by the swap.

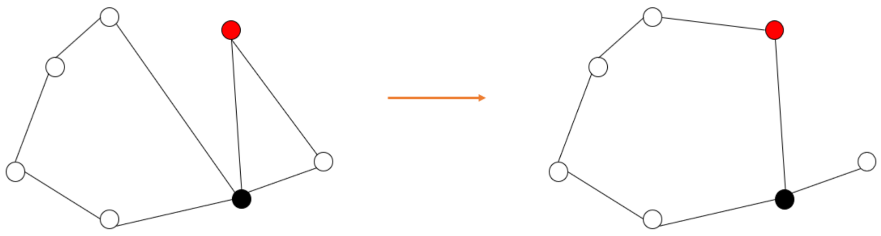


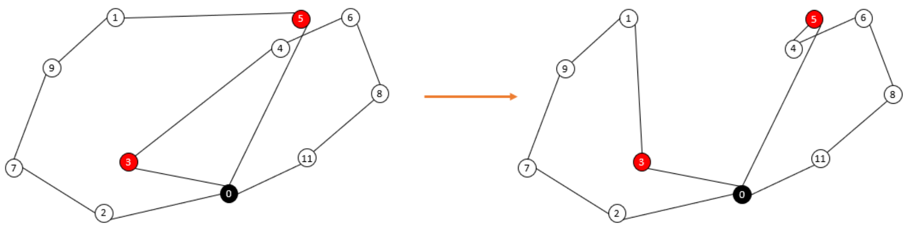
Figure 3.2: Example of routes generated by *Insertion*. The routes on the left are the two selected ones, the customer selected for the insertion is the red one and the depot is represented by the black circle. The routes obtained after a feasible insertion are the represented on the right.

It must be noticed that *Insertion* is not iteratively performed until reaching a feasible solution because is much more difficult to find a combination of routes that lead to feasibility with respect to the case of *Swap*. In fact, in our instances of *CVRP*, the customer-capacity constraint seems to be much stronger than the duration and the load constraints. This implies that *Swap* is a much easier operation since it does not change the number of customers in each route, instead *Insertion* increment the number of customers in the second route. Anyway, even if it less probable that *Insertion* would lead to a feasible insertion, it is a fundamental operation, that allows us to reduce the number of used vehicles, by trying and emptying the small routes, hence the preferential choice of small routes as the first route in the *Insertion* algorithm.

Given the feasible neighbour solution, a *Local Search* phase is performed: for each route that has been modified in the current solution to obtain its neighbour, we look for the best order to visit customers. Actually, we do not analyse all possible permutations for each route, that is done for two main reasons: we don't want to spend too much execution time in the *Local Search* step and we don't need it either, and because we want to give way to a wider exploration of the neighbourhood in the following iterations, without getting stuck in local minima.
The choice of the examined permutations is done through the following reasoning: given the customers-capacity constraint, our routes could visit from 1 to 5 customers. If the considered route has 1 or 2 customers, we don't need to evaluate its permutations, in fact with 1 costumer we have only the identity permutation and with 2 customers the clockwise and the anti-clockwise permutations are associated with the same travel cost. If we have a number of customers n that is from 3 to 5, we have $\frac{n!}{2} - 1$ permutations to consider, we do not evaluate the cost associated to the identity permutation since we have computed yet during the neighbour generation. So, for 3-customers routes we have 2 permutations to analyse, for 4-customers routes we have 11 permutations, and for 5-customers routes we have 59 permutations. Now, the exhaustive neighbourhood investigation can be done without using too much execution time for routes with 3 or 4 customers, but if we try all possible configuration also for routes with 5 customers (and a lot of routes have 5 customers) we would spend too much time in the *Local Search*, reducing significantly the number of iterations of the *CW-TS* algorithm. So we set a maximum number of permutations *num_perm* that can be tried to improve the costs of the modified routes for 5-customers routes.
After the *Local Search* phase, the neighbour solution goes through the acceptance step, supervised by the *Tabu Search* approach. Another important observation is that, since we perform the *Local Search* on the order of customers in the modified routes, the routes paths contained in the tabu list are saved as unordered sets of customers, otherwise it would be really unlikely to obtain tabu violations.

Figure 3.3 exemplifies one iteration of *CW-TS* algorithm from the neighbour generation to the acceptance step.
Finally, a summary of *CW-TS* algorithm is reported in algorithm 5.

Swap algorithm



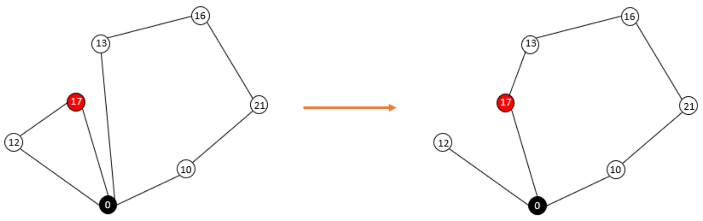
Routes before swap:
Route 1 ⟶ [0, 5, 1, 9, 7, 2, 0]
Route 2 ⟶ [0, 3, 4, 6, 8, 11, 0]

Routes after swap:
Route 1 ⟶ [0, 3, 1, 9, 7, 2, 0]
Route 2 ⟶ [0, 5, 4, 6, 8, 11, 0]

Tabu List: [{0, 4, 6, 13, 2, 1 ,0}; {0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0}]
Violate Tabu ⟶ True

Feasible swap ⟶ possible tabu moves: {0, 5, 1, 9, 7, 2, 0} and {0, 3, 4, 6, 8, 11, 0}

Insertion algorithm



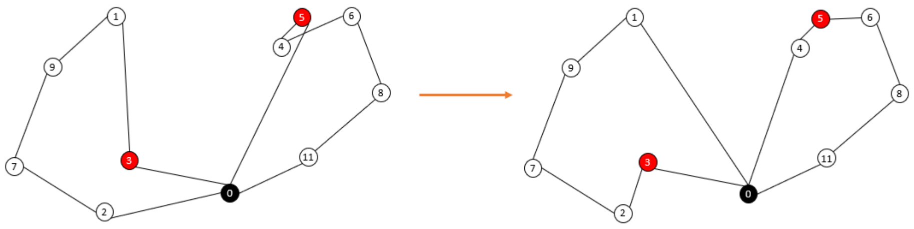
Routes before insertion:
Route 1 ⟶ [0, 12, 17, 0]
Route 2 ⟶ [0, 13, 16, 21, 10, 0]

Routes after insertion:
Route 1 ⟶ [0, 12, 0]
Route 2 ⟶ [0, 17, 13, 16, 21, 10, 0]

Tabu List: [{0, 4, 6, 13, 2, 1 ,0}; {0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0}]
Violate Tabu ⟶ True

Feasible swap and NOT feasible insertion ⟶ possible tabu moves: {0, 5, 1, 9, 7, 2, 0} and {0, 3, 4, 6, 8, 11, 0}

Local Search

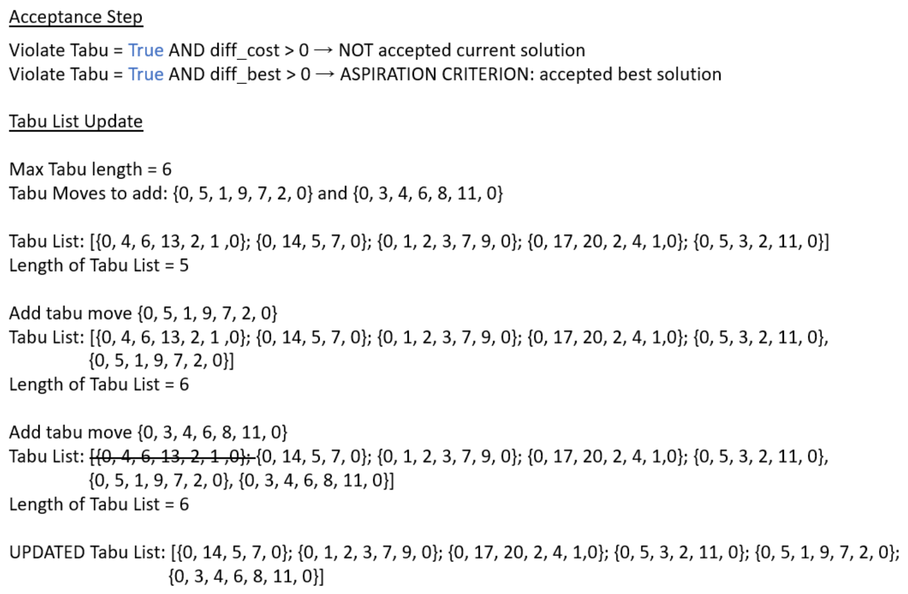


Modified routes in the neighbour:
Route 1 ⟶ [0, 3, 1, 9, 7, 2, 0]
Route 2 ⟶ [0, 5, 4, 6, 8, 11, 0]

Optimized routes:
Route 1 ⟶ [0, 1, 9, 7, 2, 3, 0]
Route 2 ⟶ [0, 4, 5, 6, 8, 11, 0]

Tabu List: [{0, 4, 6, 13, 2, 1 ,0}; {0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0}]
Violate Tabu ⟶ True
diff_cost = 15.65
diff_best = 17.34

Figure 3.3



Acceptance Step

Violate Tabu = True AND diff_cost > 0 → NOT accepted current solution
Violate Tabu = True AND diff_best > 0 → ASPIRATION CRITERION: accepted best solution

Tabu List Update

Max Tabu length = 6
Tabu Moves to add: {0, 5, 1, 9, 7, 2, 0} and {0, 3, 4, 6, 8, 11, 0}

Tabu List: [{0, 4, 6, 13, 2, 1 ,0}; {0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0}]
Length of Tabu List = 5

Add tabu move {0, 5, 1, 9, 7, 2, 0}
Tabu List: [{0, 4, 6, 13, 2, 1 ,0}; {0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0},
{0, 5, 1, 9, 7, 2, 0}]
Length of Tabu List = 6

Add tabu move {0, 3, 4, 6, 8, 11, 0}
Tabu List: ~~[{0, 4, 6, 13, 2, 1 ,0};~~ {0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0},
{0, 5, 1, 9, 7, 2, 0}, {0, 3, 4, 6, 8, 11, 0}]
Length of Tabu List = 6

UPDATED Tabu List: [{0, 14, 5, 7, 0}; {0, 1, 2, 3, 7, 9, 0}; {0, 17, 20, 2, 4, 1,0}; {0, 5, 3, 2, 11, 0}; {0, 5, 1, 9, 7, 2, 0};
{0, 3, 4, 6, 8, 11, 0}]

Figure 3.3: One iteration of *CW-TS* algorithm starts with the generation of the neighbour solution, given the current solution. The first variation is given by the *Swap* algorithm, that is iteratively performed until a feasible swap is found, once the feasibility is reached we check if the new routes violate the tabu and update the routes for the neighbour solution. Successively, we try one iteration of the *Insertion* algorithm, the result of this step does not produce for sure feasible routes: if the found routes are feasible, then the neighbour is updated and checked for tabu violation, otherwise, as in the example, the routes generated by the insertion are discarded. Then, all the modified routes in the neighbour go through the *Local Search* step that aims to reduce the travel costs of the new paths, in the example, only the routes produced by swap goes through the *Local Search*. The output of the *Local Search* contains both the optimized new routes and the reduction in travel costs associated with the neighbour solution: $diff\_cost$ is the reduction with respect to the current solution and $diff\_best$ is the reduction with respect to the best solution found so far. In the example, we have that the neighbour introduces an improvement for both best and current solution ($diff\_cost > 0$, $diff\_best > 0$), anyway the first route generated by swap corresponds to one of the unordered set contained in the tabu list (the one highlighted in blue), so the neighbour solution violates the tabu. This tabu violation implies that the neighbour solution is not accepted as a new current solution, but since it is the best solution found so far, it activates the aspiration criterion and it is accepted as the new best solution. This kind of acceptance leads to the final step that is the tabu list update: in the example, the maximum length of the tabu list is set to 6 and the current dimension of the tabu list is 5 so the first reverse move, corresponding to the set of the first old route in the current solution before swap, is inserted. After that, the tabu length is 6, so before adding the second reverse move, we have to delete the first tabu move in the tabu list. In this way, the final dimension of the tabu list is equal to the maximum allowed length.

Algorithm 5 *CW-TS algorithm*

```
1: $elapsed\_time \longleftarrow 0$
2: $S \longleftarrow$ set of selected customers
3: $initial\_feasible \longleftarrow False$
4: while not $initial\_feasible$ do
5:     Apply Clark-Wright algorithm
6:     Apply SmallRoutesElimination algorithm
7:     if number of routes $\leq$ number of available vehicles then
8:         $initial\_feasible \longleftarrow True$
9:         Save the solution: $initial\_solution$
10:    else
11:        Remove last customer in $S$
12: Set time limit: $max\_time$
13: Set number of iterations before accepting a worsening solution: $gap\_worse$
14: $current\_solution \longleftarrow initial\_solution$
15: $best\_solution \longleftarrow initial\_solution$
16: $no\_improvement \longleftarrow 0$
17: $tabu\_list \longleftarrow$ empty list
18: while $elapsed\_time \leq max\_time$ do
19:     $eliminated\_route \longleftarrow False$
20:     $violate\_tabu \longleftarrow False$
21:     $feasilble\_swap \longleftarrow False$
22:     while not $feasilble\_swap$ do
23:         Apply Swap algorithm
24:         if Swap algorithm produces feasible routes then
25:             $feasilble\_swap \longleftarrow True$
26:     Apply Insertion algorithm
27:     if Insertion algorithm eliminates a route then
28:         $eliminated\_route \longleftarrow True$
29:     Apply Local Search
30:     Check for tabu violations in the $neighbour\_solution$
31:     if at least one of the generated routes violates tabu then
32:         $violate\_tabu \longleftarrow True$
33:     Compute $diff\_cost$ and $diff\_best$
34:     if not $violate\_tabu$ and $diff\_cost \geq 0$ or $eliminated\_route$ then
35:         $current\_solution \longleftarrow neighbour\_solution$
36:         Update the $tabu\_list$
37:         $no\_improvement \longleftarrow 0$
38:         if $diff\_best \geq 0$ or $eliminated\_route$ then
39:             $best\_solution \longleftarrow neighbour\_solution$
40:     else if $diff\_best \geq 0$ or $eliminated\_route$ then
41:         $best\_solution \longleftarrow neighbour\_solution$
42:         Update the $tabu\_list$
43:         $no\_improvement \longleftarrow 0$
44:     else if not $violate\_tabu$ and $no\_improvements \geq gap\_worse$ then
45:         $current\_solution \longleftarrow neighbour\_solution$
46:         Update the $tabu\_list$
47:         $no\_improvement \longleftarrow 0$
48:     else
49:         $no\_improvement \longleftarrow no\_improvement + 1$
```

3.3 Analysis of Results

3.3.1 Python implementation

In this section, we describe the results of the *CW-TS* algorithm applied to the *CVRP* instances generated by *NP_1*. Even this part of the code is implemented in *Python* and developed using *VisualStudio Code* editor. The entire code is available in the same *GitHub* repository.
Also for this part of the thesis, we have exploited *Google Colaboratory* notepads to run multiple simulations, with the runtime type set to *None*.
We must observe that the variations, introduced by the solver of *CW-TS* algorithm, do not affect the structure of the main function in file `main.py`.
To solve the *CVRP* instances with this solver, you have to call the main function with the following command line arguments:

```
python main.py file_path -p policy -d days_simulation -s solver
```

where `file_path` is the path to the file containing the distribution of customers' demands in the considered region, in the reference code it is `grid.txt`; `policy` can be chosen among values `EP`, `DP`, `NP`, `NP_1`; `days_simulation` is the number of days in the simulation, in following results `days_simulation` is equal to `100`. Finally, `solver` is the solver used to solve the instances of CVRP and is set to `cwts`

We observe that, even if the choice for `policy` is set to `NP_1` , in order to apply the *CW-TS* solver in combination with the best policy found so far, it is compatible even with the other policies' options. Furthermore, the solver can be applied also to *CVRP* instances that are not generated by our simulation. This can be done by exploiting the object-oriented structure of the solver, initializing the solver's objects.
Finally also for this solver, we have some constant values, such as the time limit or the tabu length, to avoid magic numbers inside the code, also these values are set only once in the file `constant.py`.

For a better understanding of the code structure we report here the basic classes that are used to define and solve a *CVRP* instance: all these classes definition and their documentations can be found in the folder `Classes` of the *GitHub* repository.

```
    class Route:
    def __init__(self, cap_kg=constant.CAPACITY, cap_min=constant.TIME,
cap_cust=constant.CUSTOMER_CAPACITY):
```

it is used to define routes' objects and check capacity constraints for each vehicle.

```
    class Customer:
    def __init__(self, id, demand, service_time):
```

it defines the customers' objects and stores customers' service times, locations and demands.

```
class ClarkWrightSolver():
def __init__(self, selected_customer, depot
```

it is used to obtain a first feasible solution to *CVRP* instances.

```
class TabuSearch():
def __init__(self, initial_solution, max_time, tabu_len, gap_worse):
```

it is the core of the solver and it applies the *Tabu Search* algorithm to improve the first solution of a *CVRP* instance.

3.3.2 Previous attempts

Before reaching the final configuration of the *CW-TS* algorithm several attempts were performed: the main problems of the previous versions of this algorithm concerned the execution time to generate neighbour solutions and apply *Local Search*. In fact, those two parts are the ones more time-consuming, as evidenced by the profiling tool for *Python* software of *VS-Code*.
The reason why we want these steps to be the more efficient as possible is that we want to set a time limit of the order of seconds, since the *OR-Tools* solver takes nearly 6 seconds to solve an instance of *CVRP*. Considering that the *OR-Tools* solver is developed in *C++*, which is a compiled language, whilst our software is entirely written in *Python*, an interpreted language, we consider a good time limit for the *CVRP* solution 45 seconds. This time limit is set with the belief that if the solver for *CW-TS* algorithm was translated in *C++*, the execution time to perform the same number of iterations would be much lower, such that the *C++* time limit for *CW-TS* algorithm could be set to nearly $4-5$ seconds.

Concerning the *Local Search* step we found out that the investigation of all the possible permutations for the 5-customers routes exploited too much execution time, making the *LocalSearch* taking up to more than 30 seconds of cumulative execution time out of the 45 available seconds, resulting in a very important reduction in the number of the performed iterations.
So, a first approach to solve this kind of problem has been a parallelized version of the *Local Search* over all the possible permutations. Anyway, this kind of approach led to a critical reduction of the performed iterations, because at each iteration of *CW-TS algorithm* we had to parallelize the variables, introducing a really high-demanding operation, vanishing the positive effect of the parallelization. So, the second approach has been focused on reducing the number of examined permutations for the 5-customers routes, trying to find a compromise between the need for an accurate local search and its requested computation time.
In figure 3.4, we report the boxplots for the iterations performed on the 100 days simulation, at the varying of the number of permutations performed on 5-customers routes in the *Local Search* step. The corresponding costs, in terms of average travel costs and used vehicles, are shown in table 3.1.
Looking at the obtained results, a good number of permutations to considers is 20.

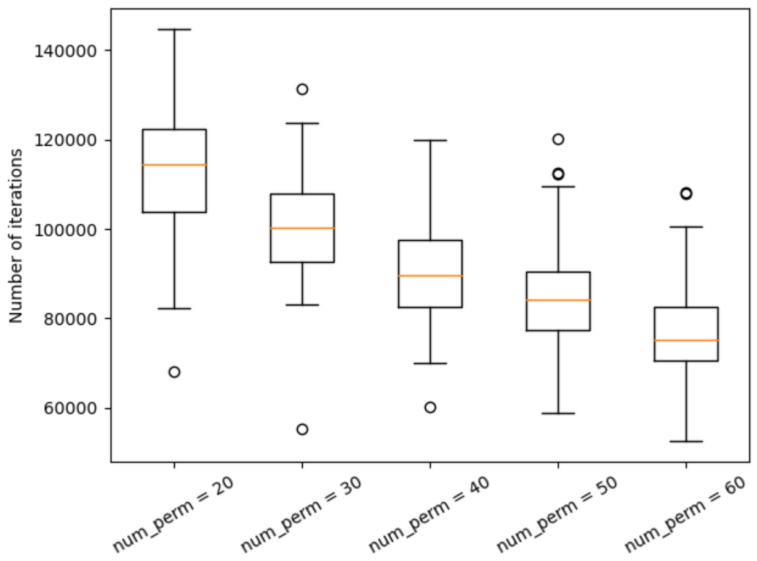


Figure 3.4: Boxplots of *CW-TS* algorithm iterations, at the varying of the maximum number of permutations of a route performed in the *Local Search* step. Each boxplot is obtained solving the instances of *CVRP* for the 100 days simulation with a time limit of 45 seconds.

	num_perm=20	num_perm=30	num_perm=40	num_perm=50	num_perm=60
Average travel cost	5367.0	5372.0	5380.0	5385.0	5383.0
Average number of vehicles	44.198	44.264	44.44	44.407	44.462

Table 3.1: Costs' variation for number of permutations in the *Local Search*, the results are based on a 100 days simulation, with time limit set to 45 seconds to solve each *CVRP* instance.

Concerning the neighbourhood generation, the following attempts have been made.

- Iterate until feasibility the *Swap* algorithm and then iterate until feasibility *Insertion* algorithm: this approach resulted in never-ending while cycles for some instances of *CVRP*, concerning the *Insertion* step. When instances allowed a finite number of iterations in both while cycles, the number of iterations of *CW-TS* algorithm were more than halved.

- Perform multiple feasible swaps: the swaps could involve more than one pair of routes and the number of swapped customers on each route could be more than one. This kind of neighbourhood generation is pretty time-demanding and requires a complicated hyper-parameter tuning on the number of routes' pairs and on the number of swapped customers.

- Un-routing and re-routing: this procedure consists of removing from the routes of the current solution some customers and then re-inserting them. This approach was useful to reduce the number of used vehicles when the *Insertion* algorithm did not have the priority for the small routes, after that priority definition, this algorithm did not improve the results anymore.

Another important step, for which we have made different attempts, is the solution initialization.
In fact, the first idea was to develop a multi-start algorithm based on *Clark-Wright* algorithm to initialize the routes, inserting a random component: the idea was to take only some of the ordered pairs of row-columns indexes in the savings matrix to merge the routes and then randomly try to aggregate the remaining pairs. Anyway, after having tried different percentages, to consider only the pairs of customers corresponding to the very highest values in the savings matrix, we found out that the best solutions were the ones that considered all the sorted elements in the upper triangular matrix of SM. So, the initialization of a feasible solution became deterministic and this multi-start approach was discarded.

3.3.3 Tuning and final results

Given the *CW-TS* algorithm definition, we have two parameters to be tuned: the number of not improving iterations before accepting a worsening solution, *gap_worse*, and the maximum size of the tabu list, *tabu_length*.
To find the best values for these parameters we analyse the performances of the *CW-TS* solver on the 100 *CVRP* instances generated with seed 57, when the applied policy is *NP_1*. The costs we are minimizing are the average daily travel costs and the average number of daily used vehicles. Figures 3.5 and 3.6 show these costs variations for different values of *gap_worse* and *tabu_length*.

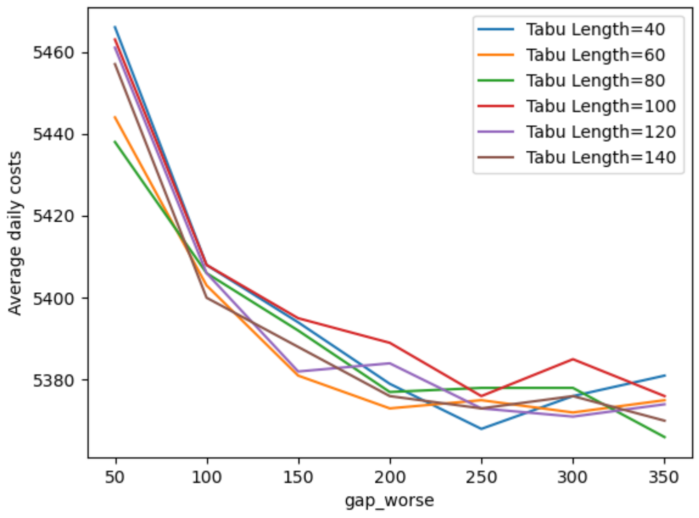


Figure 3.5: Average daily costs at the varying of *tabu_length* and *gap_worse*.

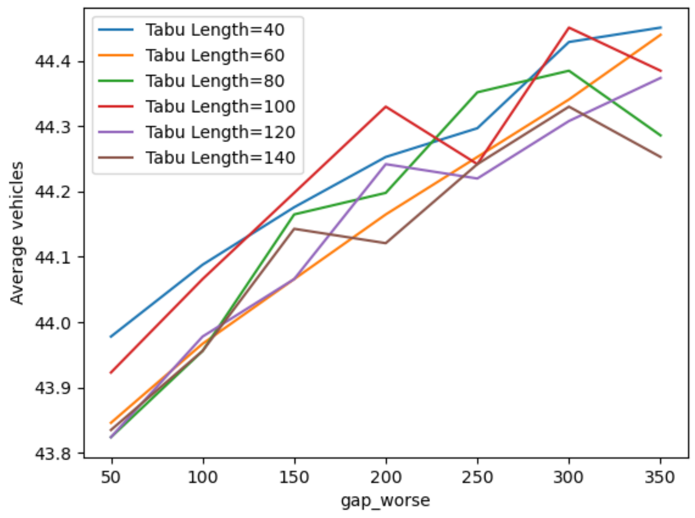


Figure 3.6: Average number of used vehicles at the varying of *tabu_length* and *gap_worse*. The variation in the number of vehicles is not much relevant since the gap from minimum value to the maximum is of nearly 0.6 vehicles.

It must be noticed that the corresponding costs, when *OR-Tools* solver is applied are

- Average daily travel cost = 5425.0
- Average number of used vehicles = 44.143

The first phase for tuning has been a grid search approach on the values

- $gap\_worse \in \{50,\ 100,\ 150,\ 200,\ 250,\ 300,\ 350\}$
- $tabu\_length \in \{40,\ 60,\ 80,\ 100,\ 120,\ 140\}$

As second step for tuning, we decided to further investigate the results obtained with $gap\_worse = 250$ and $tabu\_length \in \{30,\ 40,\ 50\}$. We obtained the results reported in table 3.2.

	tabu_length=30	tabu_length=40	tabu_length=50
Average travel cost	5384.0	5368.0	5386.0
Average number of vehicles	44.429	44.297	44.385

Table 3.2: Costs variation for *CW-TS* solver tuning, the value for *gap_worse* is set to 250.

Looking at the tuning results we decided that the best configuration for *CW-TS* solver was

- $num\_perm = 20$
- time limit of 45 seconds

- $gap\_worse = 250$
- $tabu\_length = 40$

Concerning the final improvement of the tuned *CW-TS* solver with respect to the *OR-Tools* one, we observe that the absolute improvement on the travel cost is on average of 57 *km* a day, that corresponds to percentage improvement of 1.06%. Whilst the number of the average used vehicles remains quite the same, as it is associated with a variation of 0.154 vehicles.

To further evaluate the possible advantages of using *OR-Tools* solver or *CW-TS* solver, we consider the loads and the durations of the single routes found during *CVRP* daily optimizations. We report the boxplots comparing the two solvers results in figure 3.7: as can be noticed *CW-TS* solver produces more balanced routes, in fact the variance of the routes' loads and especially of the routes' durations is lower. These results can be really important for the planning of the routes since it allows to plan more balanced work shifts.

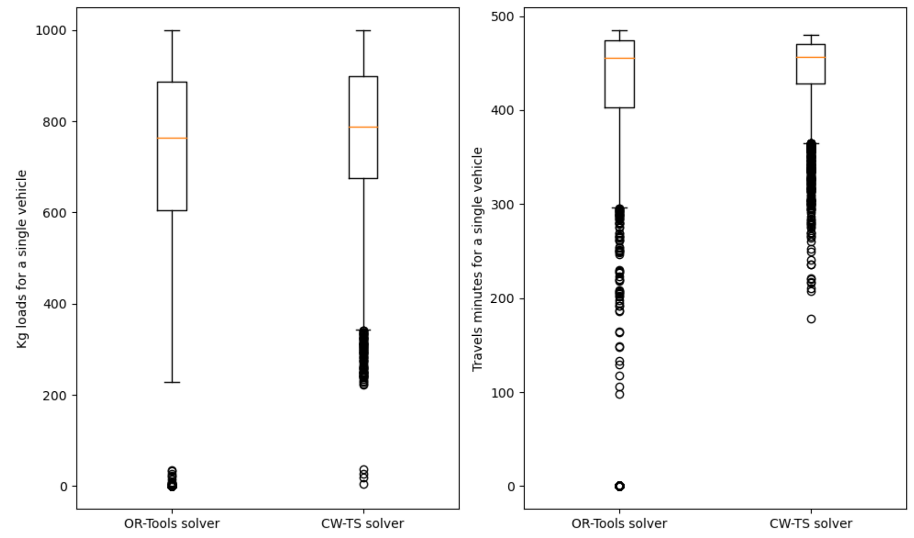


Figure 3.7: Boxplots comparison for the characteristics of the routes found with *OR-Tools* and *CW-TS* solvers: on the left the comparison of the average load in *kg* for the singles routes, on the right the boxplots of the durations expressed in *min*.

Finally, we propose a comparison of the two solvers' results using different seeds for the daily *CVRPs* scenarios' simulations: as can be noticed in table 3.3, the tuned version of the *CW-TS* solver outperforms the *OR-Tools* solver also on these scenarios and leads to an average reduction in the travel lengths of 47.2 *km* a day.

	vehicles *OR-Tools*	vehicles *CW-TS*	cost *OR-Tools*	cost *CW-TS*	improvement (km)
seed = 43	43.901	43.956	5387.0	5334.0	53
seed = 2	44.176	44.319	5424.0	5378.0	46
seed = 89	44.297	44.418	5444.0	5398.0	46
seed = 1074	43.901	44.044	5355.0	5316.0	39
seed = 551	43.824	43.912	5421.0	5369.0	52

Table 3.3: Comparison of the results obtained with *OR-Tools* and *CW-TS* solvers using different seeds for scenarios' simulations. The average reduction for the daily routes, when applying *CW-TS* solver is of 47.2 km a day, in the simulated scenarios.

For the sake of coherence, we propose a last comparison that highlights the costs' reduction due to to both the application of *NP_1* policy and the *CW-TS* solver: the combination of both these optimization approaches reduces the costs for the deliveries of about 3.40%.

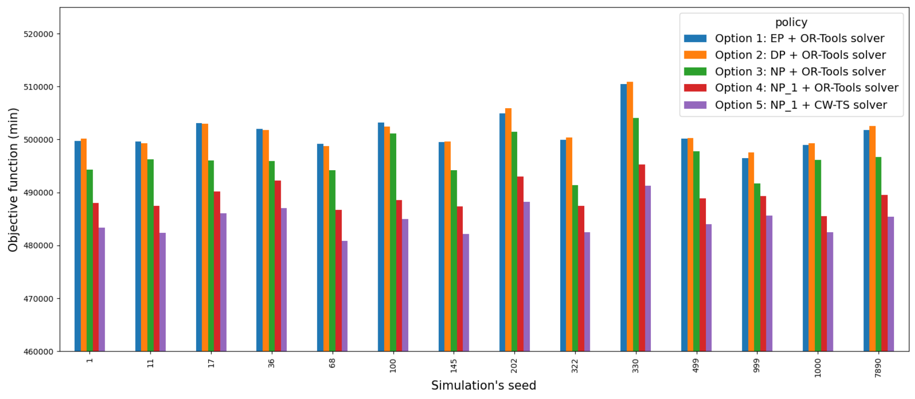


Figure 3.8: Total travel costs for datasets generated with different seeds. The simulation is made of 100 days and the costs sum the travel costs of all the vehicles for all the days.

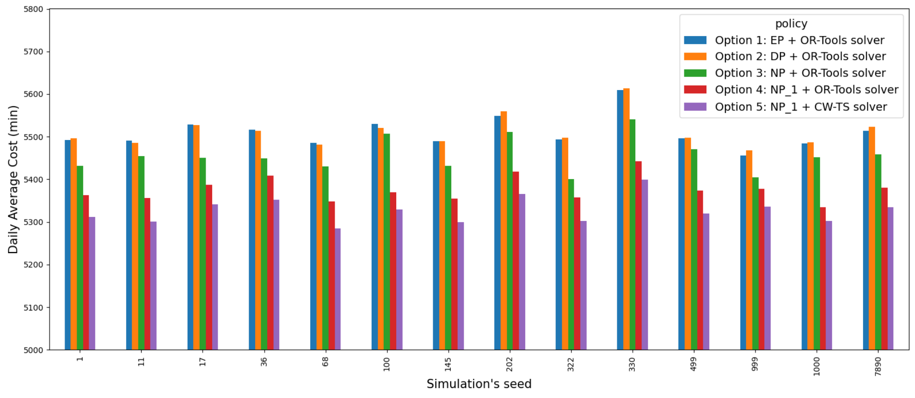


Figure 3.9: Average daily travel costs for datasets generated with different seeds. The average takes into account the travel costs associated with each daily *CVRP*.

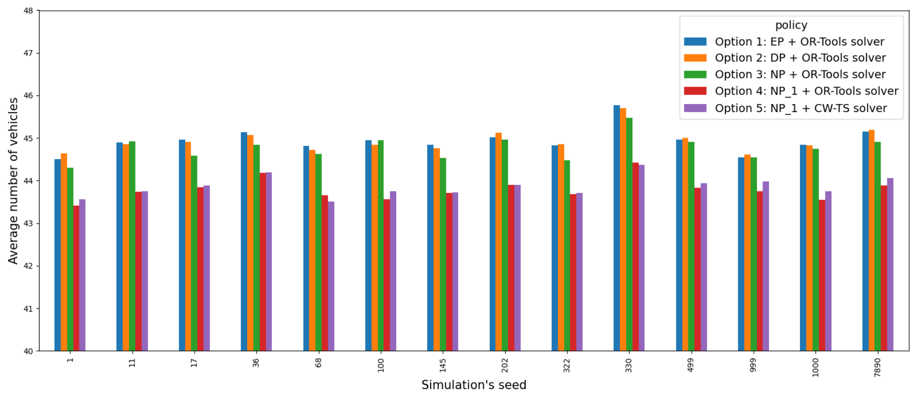


Figure 3.10: Average number of daily used vehicles for datasets generated with different seeds. The number of vehicles is averaged over the simulation's days, considering each day the number of vehicles to solve the *CVRP*.

Chapter 4

Future Developments

The work done in this thesis offers some starting points for future developments that can be investigated both as a subject of a future thesis and as an application to real-world datasets.
A first approach for future development concerns the translation of this thesis' code into a compiled language, such as *Java* or *C++*: this translation could reduce significantly the runtime for the *CW-TS* solver. Moreover, reducing the execution time of the solver could open the way to the implementation of more complicated neighbourhood functions, to further improve the *CVRP* solutions. Another improvement concerning the solver could be the development of other initialization functions to obtain the first feasible solutions to the *CVRP* instances: this way a parallel multi-start algorithm could be implemented. Concerning the parallelization of the algorithm, *C++* language is suggested, since the parallel implementation can be done very easily.

About the policies for customers' selection, some other policies could be defined and this is an open field that can be deeply investigated. A more defined development concerns the definition of the distributions used to compute the indexes for the policies *NP* and *NP_1*: first of all the parameters of the distributions could be estimated from real-world data, by means of statistical approaches, then the customers' arrivals could be modelled by a Poisson distribution, instead that using an integer Uniform. The use of a Poisson distribution on real-world data could lead to interesting results for the furniture company and it is a distribution that is highly used to model arrivals over time, given the hypothesis of independent arrivals over non-overlapping time-windows. Instead, the multinomial distribution over the region of interest could be maintained, applying the splitting property on the Poisson distribution to assign the customers to the corresponding cells.

Chapter 5

Conclusion

To sum up, the two main targets of this thesis have been reached: we found out a policy for the customers' selection task that exploits the future stochastic information in an appropriate way: applying policy *NP_1* we were able to both reduce travels costs, to the benefit of the furniture company, and satisfy customers' demands, without producing postponed orders. The comparison to the naïve policies, *EP* and *DP* highlighted an average reduction in the costs of 2.44% over different scenarios simulations.

The development of the *CW-TS* solver opens the way to further costs reduction. The improvement obtained by solving the *CVRP* instances using our solver instead of the *OR-Tools* one is of about 0.96% and the obtained routes allow a more balanced plan of the work shifts. This improvement comes at the cost of an execution time that is nearly 7 times higher, but could be hopefully reduced by an adequate translation in a faster programming language, such as *C++*.
Finally, the considered constraints for the *CW-TS* solver are the ones required by our case study, but other constraints, to meet other requirements, could be easily added without much effort.

Acknowledgement

Per concludere vorrei ringraziare le persone che mi hanno supportato nel mio percorso universitario, soprattutto in questi ultimi due anni per poter raggiungere il traguardo della Laurea Magistrale.

Innanzitutto, vorrei ringraziare la mia famiglia, che mi ha permesso di affrontare i miei studi, fornendomi tutto l'appoggio e l'aiuto che mi serviva per poter ottenere i migliori risultati.

Ringrazio inoltre i miei colleghi e amici che ho conosciuto in questi anni, che hanno reso la frequenza delle lezioni non solo un momento di apprendimento, ma anche un momento di amicizia che è rimasta anche al di là dell'ambito accademico. Un particolare ringraziamento è per Valeria, mia collega e soprattutto amica con la quale ho affrontato svariate situazioni, positive e negative, uscendone sempre con ottimi esiti.

Infine, un sentito ringraziamento alle mie amiche di lunga data, Giulia, Valentina e Dahiana con le quali ho mantenuto un bellissimo rapporto di amicizia in tutti questi anni e che hanno sempre apprezzato il mio impegno, incoraggiandomi a puntare sempre al massimo.

Bibliography

[1] Maria Albareda-Sambola, Elena Fernández, and Gilbert Laporte. The dynamic multiperiod vehicle routing problem with probabilistic information. *Computers & Operations Research*, 48:31–39, 2014.

[2] Paolo Brandimarte and Giulio Zotteri. *Introduction to distribution logistics*, volume 21. John Wiley & Sons, 2007.

[3] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[4] Laurent Perron and Vincent Furnon. Or-tools.

[5] Paolo Toth and Daniele Vigo. *The vehicle routing problem.* SIAM, 2002.

[6] Nikolaj van Omme, Laurent Perron, and Vincent Furnon. or-tools user's manual. Technical report, Google, 2014.

[7] Eleonora Vardé. A top-down approach for the dynamic vehicle routing problem. Master's thesis, Politecnico di Torino, 2018.

[8] Min Wen, Jean-François Cordeau, Gilbert Laporte, and Jesper Larsen. The dynamic multi-period vehicle routing problem. *Computers & Operations Research*, 37(9):1615–1623, 2010.