POLITECNICO DI TORINO

Corso di Laurea Magistrale

In Ingegneria Gestionale

Tesi di Laurea Magistrale

Dynamic and stochastic vehicle routing problem: a simulation tool for a large-scale study case

Relatore

Prof. Arianna ALFIERI

Candidato

Marco PAPPALARDO

Anno Accademico 2020/2021

Contents

List of Figures

List of Tables

Abstract

Logistic management is an increasingly topical issue nowadays. Most enterprise try to improve their economics by optimizing their supply chain, continuously reducing costs in the face of improving performance. The current market is mostly made up of small and medium-sized enterprises (SMEs) not used to thinking in term of global optimum and often do not even have the internal skills necessary to make their system efficient. Over the past twenty years globalization has destroyed the boundaries of local markets, as well the single European market brought to overcome the traditional vision of "local-market". Over the past twenty years internet has totally changed consumers and their consumer habits. The e-commerce, born as an opportunity is now a consolidated reality. Some large companies have begun to look abroad with a view to growth, others in order to survive in saturated national markets. All this has resulted in several changes in these company: new strategies, need for new management, birth of new roles. Logistics have become the key to success in industries, supply chain management a key function in order to minimize cost or maximize profits.

In this work we will discuss a technical problem related to a famous logistic theme,

the vehicle routing problem (VRP) in one practical exception. The purpose is to demonstrate which and how many benefits are brought by an engineering approach to problems.

Let's imagine a small furniture shop with few employees and not-owned furniture factory. The shop decide to sell online using a proprietary website. The demand is low and shipments are managed by the shop itself. For this type of products is needed an attended delivery, in other worlds the customer has to be at home the day of the delivery. This small shop promise to customers a due date by which they will receive the goods bought. Moreover the furniture shop has not an ERP system to manage neither employees or production. This situation is common to small reality due to cost and human skills lack. The shop is based in Turin and usually the customers come from the city or from the surrounding areas. However since the shop decide to sell from their e-commerce platform now receives orders (let's assume) from the entire North Italy. This growing market brought a totally new level of complexity for the daily operations of the shop, historically shipments were not a problem and were organized manually from an employee but with a growing demand it became impossible.

A simulation tool will be developed in order to represent in a realistic way this scenario, and then solve it. Moreover to address the problem in a proper way will be developed a function able to optimized the delivery to customers of the shop.

Chapter 1

Introduction

1.1 Motivation

Supply chain management and logistic are becoming increasingly topical issue in contemporary world. When we talk about logistic we include the entire management of resource from planning to final delivery. Today an efficient logistic management can be a success factor for a company, without considering aspects related to employment and the growing attention to polluting emission of which transport is one of the main causes. Furthermore the cost of logistic is sunk, the customer is unwilling to pay for a service of which he does not see the direct result. So there are several reason why the topic of transport generates more and more interest. In this work we will focus on the last part of the supply chain, the final shipment of goods to customers. However we will not discuss a last mile logistic problem because our delivery will take place far from the depot, in other word we will address a vehicle routing

problem on a large geographical scale. The problem assumes greater importance if contextualised in the panorama of SMEs. In the final section of this chapter will be clarify the specific situation introducing a study case scenario. But first we need to go deep into the phenomenon of SMEs and their logistic management.

1.2 A short SMEs overview

According to the European Commission standards (further information in EU recommendation 2003/361) in Italy around the 79,5% of business are micro-enterprises, than 18,2% are small, finally 2,3% of Italian company are medium and large companies (data source: Istat).

Looking to Italy we can say that around the 98% of existing economic realities fall into the category of SMEs; broadening the gaze to the whole of Europe we find the same situation with the 99% of all business being small and medium-sized enterprises.

To better understand what we are talk about, table 1.1 allows us to understand according to which criteria companies are classified.

Company category	**Staff**	**Turnover**	**Balance sheet**	**% of italian companies**
Medium and big-sized	≤ 250	$\leq 50k€$	$\leq 43k€$	2,3 %
Small	≤ 50	$\leq 10k€$	$\leq 10k€$	18,2 %
Micro	$\leq 10 \geq 4$	$\leq 2k€$	$\leq 2k€$	79,5 %

Table 1.1: SMEs classification, data source Eurostat

The economic impact of SMEs companies should not only be seen in relation to the country's GDP, but must be correlated with the supply chain in which they operate. This first consideration brings to the center of our attention the logistic theme that will be dominant from now on.

A main trend of every market is the e-commerce. To survive a SMEs business cannot ignore strong changes in the market but often the lack of suitable human resource and myopia in recognized the need to innovate leads to change when it is too late.

Before delving into a more deep look at e-commerce phenomenon, take note that COVID-19 drastically changed quite all industries, has promotes changes in customers behaviours and at the same time it has resulted in the need for a quick response from companies.

1.3 SMEs choices about logistic outsourcing

The first answer of many small business to the Coronavirus was investing in e-commerce solution. To compensate for lost sales in physical store, we have observed an exponential increase in online sales with home delivery .

From the Eurostat database we can observe the dimension of the growth in numerical terms.

	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Italy	5	5	6	8	8	10	11	13	14	14
EU(28 members)	15	15	16	17	18	19	20	20	20	20

Table 1.2: Enterprises with e-commerce sales (% of enterprises, all enterprises)

	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Italy	10	15	17	20	22	26	29	32	36	38
EU(28 members)	40	42	44	47	50	53	55	57	60	63

Table 1.3: Internet purchase by individuals (% of individuals, last online purchase within 12 months)

The information reported above shows an important growth in e-commerce until 2019, which has however surged during pandemic period. From a recent market survey carried out by the Italian Consortium of Electronic Commerce, Netcomm, it emerges that by the end of 2020 the sector will have a further growth of 55% in the number of consumers who shop online.

Actually local business are able to use many sales channels, some of them are incorporated in popular social apps others require affiliation with established online sales platform (i.e. eBay and Amazon). If in the second case they became customers themselves and must be subjected to conditions that are not very negotiable, a lots of them prefer to use the web to reach customers and manage the logistical aspects internally.

Owned e-commerce site offers any business significant growth potential, especially by opening up to physically unreachable customers, but on the other hand

the large marketplaces mentioned above make more difficult to create a own slice of loyal customers. Moreover customer loyalty strongly depends on the service offered including the entire customer experience: from the accessibility of the website to the punctuality in the delivery. The customer is not interested in logistics management, which is a deep-rooted aspect of his experience, however his expectation are highly dependent on the consequences of this activity. Moreover managing an online business involves the need to manage a web reputation, which involves exposing oneself to negative reviews.

From the foregoing consideration it may appear evident that logistics plays a key role therefore it can be considered a reasonable choice for SMEs to outsource the logistics to 3PL provider. A research by Andrea Payaro (Vice President SCM Academy) and Anna Rita Papa turns out that outsourcing the logistics management is a controversial choice for small business. The main reason is the belief that the benefits brought by outsourcing would not bring strategic advantages linked to a reduction in cost and a higher level of service, however this phenomenon can be tracked back to some factors, of which I report a brief summary:

- SMEs are not able to estimate the costs of logistics. By their nature they do not have an administrative structure that takes note of the main cost centers making it impossible to quantify the opportunity cost of switching;

- SMEs do not know the range of operations that a logistic provider is able to accomplish, often considering only warehouse management and shipping as "logistic". They ignore the optimization approach that a 3PL operator can adopt in management, also in terms of IT solutions for traceability and

identification of goods;

- SMEs often fear a losing of internal skills and a more complex decision-making process.

The reticence to innovate, the delay in arriving on the market caused by ignoring new trends is a very widespread problem sometimes with fatal consequences even on large companies with consolidated economics (take a look at Kodak case). Obviously the impact on small business are even more fatal.

1.4 Presentation of the problem studied

In the abstract we mentioned the specific case that led to this paper: a small furniture shop with no intention of leaning on an external 3PL provider. This is now a far more credible case. You can imagine a small business, few employees, no-owned factory, no ERP system. This could be a familiar business heavily stressed by economic condition.

The furniture shop usually received orders by surroundings, but recently decided to have his own webstore to expand its clientele to a much higher geographical area. It has always been the owner to manage the shipments of the goods, the experience he has accumulated over the years allowed him to comfortably serve customers while remaining within the promised deadline for delivery. The shop use its owned van to delivery. However after the implementation of the webstore deliveries where no longer in range of dozens of kilometres, having to take place in towns and cities in other regions. The first attempts to manage this increasing complexity consisted

in continuing to manually manage the scheduling of deliveries. This approach was obviously ineffective, in particular the cost of travel costs were such as to offset the increase in demand and turnover. The main problem was that some customer were in extremely isolated and distance places, and sometimes to avoid a delay in delivery the van was sent to accomplish a single order. However this choice generated chain delays to other pending orders, ready for delivery but waiting in the warehouse. Furniture deliveries must take place in presence of the customer, so they must be agreed in advance. Furthermore promising a date and not meeting it generates strong discontent with the customer and destroy the reputation of the store. The shop owner, who is not an engineer, makes his shipments decisions based on the vehicle capacity, the delivery due date of the orders and an estimate of the time that certain deliveries will require; on the basis of his estimates he often agrees on deliveries for "tomorrow" which then do not take place. In low demand periods, the owner manage to meet the deadlines promised to customers; but in cases of high demand, delays become a daily situation.

We can rephrase the problem by asking: how the owner should manage isolated and distance order? Maybe he could wait for other orders to appear close to the most problematic ones, but how long to wait and how to consider two orders "close"?

We will try to give an answer to these and consequential question over the next chapters. The shop chose to manage logistic aspects internally, but this should not go beyond trying to optimize outgoing flows of goods. To solve the problem we will create a tool capable of introducing measurable improvements to shipments with the aim of managing transport costs and avoiding delays with respects to the promised

deadlines. The key aspect will be the logic within the next order to be served must be chosen.

1.5 Organization of the work

The next section are organized as follows.

- Chapter 1 introduce the problem giving a first overview to the scenario developed in the present dissertation.
- Chapter 2 is a literature review. It introduce the theorical definition of vehicle routing problem explaining its main variants developed during the last decades. In particular more emphasis is given to stochastic and dynamic VRP as well as to some publications that have most influenced the present work.
- Chapter 3 provide a wide description of the study case. Firstly the problem is contextualized according to literature lexicon, then the problem structure is formalized using an algorithmic approach. After the presentation of the main assumptions needed to develop the problem, it is introduced the simulation tool, the software created to address the problem in a realistic way.
- Chapter 4 is entirely focused on the likelihood function and its implementation in the software. The main theorical contribution of this paper is firstly presented with a simple example and then made complex by using it in a realistic scenario. The likelihood function is finally extended to the graph representation.

- Chapter 5 show some simulation output obtained from various run with different parameters settings. This make possible to analyse strengths and weakness of the tool.
- Chapter 6 is the brief conclusion of the work. Here some future perspectives are proposed.

Chapter 2

Literature review

The Vehicle Routing Problem (VRP) is a common theme addressed in operational research. It consist in manage a fleet of vehicles in order to accomplish several deliveries within a given space, often with additional constraints. In this work is studied a problem associated to VRP, but in a practical way which distances it from classical literature; in most cases it is studied as a MILP (mixed integer linear programming) problem, however, if this path has been followed in the present work it would have been difficult to make a real contribution to current literature. Given the mathematical complexity of recent papers above VRPs, this university thesis could not have approached the issue in a dignified way. Instead the approach that will be followed aims to avoid most of the theorical assumptions introduced in VRPs, i.e. a graph will not be used to represent customers locations but the real road network. We will look at a plausible problem, analyze it with an engineering eye, study a possible solution, implement it and evaluate the results. In chapter 3 there is an

explanation for these choices. This more practical method of dealing with VRPs is not very present in the literature, except in a few cases, therefore the marginal benefit brought by this work can open to a different point of view on this issue as well as making it an effective contribution.

Despite this, the starting point cannot not be an analysis of literature. After a quick look at the first VRP formulation, will be discussed the literature evolution around the theme, and a classification of VRPs main features will be introduced. Finally some reference to papers that are the main inspiration for this work.

2.1 Capacitated vehicle routing problem

Vehicle routing problem was first introduced by Dantzig and Ramser (1959), their main goal was to model a fleet of homogeneous truck to serve a certain number of gas station looking minimizing the total travel distance. Only later, Clarke and Wright (1964) formalized the generic VRP as a linear optimization problem. The key features were a generic set of known customers (geographically dispersed) with specific demands , a central depot as a starting point for tours and a fleet of vehicle characterized by different loads capacity. Customers (and depot) are modelled as nodes of a graph, representation from which it is possible to derive a distance matrix, input for the linear optimization problem. Hence the optimization model tried to assign customers to vehicles and sequences the customers visited by each vehicle in order to minimize the total routing cost. Practically the output is an ordered set of customers to be served by each vehicle. This classical version is also known as

Capacitated Vehicle Routing Problem (CVRP).

$$min \sum_i \sum_j \sum_k d_{ij} x^k_{ij} \tag{2.1}$$

$$st. \sum_i x^k_{ij} - \sum_i x^k_{ij} = 0 \tag{2.2}$$

$$\sum_i \sum_k x^k_{ij} = 1 \tag{2.3}$$

$$\sum_i x^k_{i0} = 1 \tag{2.4}$$

$$\sum_j x^k_{0j} = 1 \tag{2.5}$$

$$\sum_i \sum_j q_i x^k_{ij} \leq Q \tag{2.6}$$

The most simple case of VRP is the traveling salesman problem (TSP) where the objective is to sequence a set of geographically dispersed customer while minimizing the total travel cost, compared to CVRP this is a single-vehicle case.

2.2 Evolution of VRPs

Over decades the problem has become more popular and the number of papers has increased dramatically (a more accurate view of this phenomenon is described in Psaraftis, Wen, Kontovas, 2016), the raising complexity of modern industry resulted in the need to help decision making processes specially in logistics or supply chain management field. Obviously the first VRPs formulations lacked of realism, which involved the need for multiple assumption. During last decades technological ad-

vancements and dozens of papers about the topic have brought academic literature and real-life applications closer together. Actually many company uses VRP software to better manage the cost structures, and the VRP is one of the most interesting problem in Operation Research.

Some assumption have been relaxed to better reflect the reality of operations, allowing the development of new variants of VRP to incorporate time windows for pickup and delivery, temporal dependencies on travel times (e.g. due to traffic congestion), multiple commodities, stochastic customer habits and many others. Lenstra and Rinnooy Kan (1981) shown the NP-hard problem nature of VRP, hence the multiple constraints introduced by the variants to the main version have raised the level of computational complexity making exact algorithm not suitable with large problem instances. Software for real application implement heuristic and meta-heuristic solution approach as well large as large-scale problems. According to these consideration, over the years various authors have proposed multiple solution approaches to the multiplicity of versions. Several surveys classified VRPs literature according to different perspectives. However, this task has not made easy by the copious production of new publications. At the same time there are further problems: taxonomy may vary a lot in different papers and work on a common benchmark is difficult since the difference in structuring the problem (this case is especially true in dynamic VRPs, no benchmark instances are available).

For the purpose of this thesis work we can at first refer to Braekers et al. (2016), a less recent overview which however helps to understand the reason of the next classification. This paper does not consider any combined problems that will be in-

troduced later, but gave an interesting point of view around the "future" trends of VRPs studies and allows to better contextualize the continuation. The authors used a taxonomic framework to evaluate the relevant literature published between 2009 and 2015 relying on five categories: type of study, scenario characteristics, problem physical characteristics, information characteristics, data characteristics. Some consideration highlights the benefits that this work can give to the current literature. What emerges from the analysis is a low utilization of a simulation approach while metaheuristics one seem to be widely preferred, information are quite often assumed deterministic and only in a few cases are stochastic or unknown. Moreover when stochasticity is considered it concerns only one aspect of the problem. Multi-period problem have hardly been dealt with, even more poorly those that incorporated stochastic aspects. Our study case considers both dynamicity (so a multi period problem) and stochasticity in customer behaviour, demand and geographical location.

A more recent review is Mor, Speranza (2020), here the focus is set on the type of decision to be taken to solve the problems. To better explain: the classical VRP only require to assign customers to vehicles and to sequence them. A VRP whit profits require to choice which customer to serve, a VRP with time windows (VRPTW) requires to compliance with this temporal constraint. The authors aim to better deal with the most recent variants of VRPs according to their multi-constrained nature, element derived from the need for each job to get closer to real-life applications. This classification begins from the classical CVRP (assignment and sequencing) and follow to:

- VRP *with profits*, when making a decision about which customer to serve according to maximize profits;

- VRP *with slip deliveries* (SDVRP), which adds the choice of the quantity to be delivered to a customer allowing a fractional delivery;

- VRP *with multiple commodities*; usually is considered a common metrics as volume or weight to represent different goods, however in some application may be relevant to consider additional constraint to distinguish goods, i.e. in multi-depot scenario which one preserves a type of good or in presence of an heterogeneous fleet with restrictions on transportable goods;

- VRP *over time* are described as dynamic problems where the starting time of each route is to be decide by the model.

This last category help us to introduce a larger set of VRP variants that consist of combined problems; these constrain the resolution of a VRP to further constraints not strictly connected to routing. The key element is a time-horizon discretized in finite periods, in each a vehicle could perform a tour. The usual period is a day, so when to serve a customer became a decision to be taken "over time".

2.3 VRPs features classification

This work has no intention to provide a new classification, however, to the best of our knowledge, a brief discussion of the main variants of VRP follows. This lengthening is necessary for the reader to recognize the lexicon that will be used in the next chapter;

it will also help to recognize the specific characteristic that will be introduced into our problem.

To get a more complete overview of what follows it is possible to refer to Psaraftis et al. (2016), due is a similar but more in-depth analysis. The nature of a VRP can be static or dynamic, the type of information can be deterministic or stochastic. Combining these feature we obtain four main variations:

- SD VRP *static and deterministic* vehicle routing problem
- SS VRP *static and stochastic* vehicle routing problem
- DD VRP *dynamic and deterministic* vehicle routing problem
- DS VRP *dynamic and stochastic* vehicle routing problem

We can define a dynamic problem when not all the information are available from the first moment and some of them become known only over time; in contrast a static problem has all the information from the beginning. Information can be deterministic when input are known without uncertainty, stochastic when are affected by some form of randomness and therefore when are known only in generic terms of probability distribution or not known at all. Stochasticity can be present on one or more pieces of information, in the most common cases it concerns customer demand or location. A more recent and complex case are stochastic travel times; according to Ritzinger et al. (2016) they can be modelled as time-dependent, stochastic or both. An interesting development particularly suitable for making last mile logistic problems more realistic.

Assuming these types of VRPs we can identify other distinctive elements more connected to the each element of the problem.

- **Vehicles**:
  - *fleet size* can range from one vehicle to infinity, this last case is extremely rare;
  - *capacity constraints* are commonly adopted except in rare cases where the volume of goods makes this constraint negligible;
  - *specific vehicle required* for a specific good type.
- **Customers**:
  - *stochastic location and demand* as suggested above;
  - *possibility to refuse a delivery* (another form of stochasticity);
  - *release or/and due date* constraints;
  - *attended delivery*: the customer must be present, therefore the time of delivery must be agreed in advance
  - *time window constraints (TW)* so specific period during which delivery is possible:
    - *soft TW* allow forms of earliness or tardiness;
    - *hard TW* (less adopted) can be realistic in a few scenarios, but depending on other factors as fleet size and capacity can make problem instances infeasible;

  - *type of service* to customer:
    - *delivery-only* or *pick-up only*;
    - *delivery and pick-up* combined.

- **Tours** can be characterized by:
  - *single or multi-depot* scenarios, with possible constraints of departure and return on each vehicle;
  - *open or closed routes*, i.e. in the so called Open VRP (OVRP) vehicles do not have to return to the depot as the final point of their tours;
  - *max route duration or length*;
  - other constraint can be introduced in specific cases, i.e. forbidden arcs on a multi-graph.

Finally the objective function can minimize distance or travel time, maximize profits or the number of customers served etc... In any case a proper formulation depends strongly on the available data and the purpose of the solution strategy.

2.4 Combined VRPs

Nowadays a lots of organization based their organizational structure around the key function of the supply chain management. Literature responded to this change developing more complex variants of VRPs that combine the distribution of goods to other business needs. Typically these are dynamic problems with the possibility to add a stochastic element.

Inventory routing problems (IRPs) add the inventory management to the standard VRP. The basic version was introduced by Archetti et al. (2007) and present multiple retailer each with a minimum and maximum inventory level, and a supplier with a variable quantity available. The problem is dynamic, in each period the system have to decide which customer to serve according to stocks availability. The objective function point to minimize inventory and routing cost. Recently stochastic aspects were added to the IRP often concerning customers demand. In Coelho et al. (2014) it is proven that considering this aspect increase the quality of solution.

Periodic routing problems (PRPs) are characterized from having to visit customer multiple times during a planning horizon. Often this problem is associated to a supplier-to-retailer replenishment scenarios, and it is common to assign a fixed delivery frequency to customers. A basic formulation of PRP is provided in Campbell et al. (2014). In Archetti et al. (2015) is introduced an interesting variant, a multi-depot VRP with release and due dates.

The family of VRPs with release date as well can be considered combined problems. These dynamic problems involve deciding when to serve a customer. Given a set of existing customer the decision maker has to choose if start a route or wait for new customer to appear, wait can bring to build better solution.

A more wide overview around integrated variants is provided in Vidal et al. (2020) where a classification is developed according to the kind of combined problems to solve. Main decision elements are:

- *routing and districting*, where a geographical partitioning is required;
- *routing and facility location*, where the purpose is to choice the place for a

facility comparing routing cost in different cases;

- *routing and fleet composition*, in order to dimensioning the vehicle fleet based on variable demand pattern;

- *routing, inventory, production management*, is a development of IRPs whit the production planning problem considered as well.

A general consideration regards recent trends, most of the literatures problems are addressed to solve last mile logistic scenarios, long-range routing problem represent a minority. The authors often conclude their papers suggesting a more real approach for further development both considering stochastic information (overall in travel times) and real road network.

2.5 Main references

A few articles has been the main inspiration for this work.

In Han et al. (2017) is presented a single-depot vehicle routing problem with time windows (SDVRPTW) with attended home delivery (AHD); the purpose of this paper was to analyse the appointment scheduling problem with a stochastic behaviour in response time to the courier appointment. In the moment a delivery happens the courier intercoms and awaits the arrival of the customer, here are two type of scenario: customer do not respond (no-show), customer respond with random response time. In courier's tight scheduling that source of randomness can generate delays in subsequent deliveries. The authors addressed the problem in the broader

framework of VRP combining: allocation of customers to vehicles, sequencing the tour of each vehicle, determination a dynamical start and maximum waiting time at each customer based on the previous delivery ending time. The comparison between two type of approach, hierarchical and integrative, showed better performances in the second one.

A Decision Support System (DSS) for attended home delivery was instead developed by Bruck et al. (2020). This paper explains the design and implementation of an IT support system for Iren Group, an Italian company operating in energy sector. The specific problem was to manage the dispatching of technicians to customers locations. This scenario is different from our case study because of the problem structure. Here each municipality of some regions have their own team technicians (multi-depot), who mainly work in their geographical area of competence, with the possibility of sending them to nearby area in high demand cases. Customers make appointments from an online portal that is constantly updated to show the time slot available to receive the technician's visit. In this case the scheduling is not done by the system. The key problem is the allocation to resources and the sequencing of location to visit. The simulation showed significant improvements in service quality compared to the previous manual scheduling adopted by Iren; this software is currently being used. Moreover this DSS "run" on the real road network, the API serviced used by the author is the same implemented in the tool developed in this thesis.

The idea to implement a practical tool born within the analysis of this paper. However we do not have historical data to compare performance and this is the

biggest weakness of this work. To overcome this problem a lot of input parameters will be treated as experimental parameters with the aim to understanding in which situation the tool performs better and in which it is not suitable.

2.6 Our contribution to literature

The main contribution this work is supposed to provide is answer some of the rising questions in the current literature. Facing a dynamic and stochastic problem with release and due dates from a theorical perspective would not make this a significant paper given the large and complex literature already present, to get around this I create a realistic case that included the least addressed features as long-range routing, stochastic customer behaviour, real road network. Moreover is proposed a simulation approach consisting in an ad-hoc developed software. Nevertheless a theorical contribution is represented by a new choice function to select cluster of customers. This function is thought to solve the trade-off about waiting new customer to appear or starting a route in presence of stochastic customer behaviour and for the rare exception of long-range routing scenario.

Chapter 3

Problem definition and simulation tool

In section 1.4 we introduced our study case with a qualitative description. In this chapter we will deeply discuss any aspect of the scenario starting by contextualizing it according to the literature taxonomy, then restructuring the problem based on an algorithmic logic and explaining all the assumption done in implementation. Finally will be presented the developed tool and will be showed some example of input generated and output expected.

A brief reminder of the furniture shop case study. The main problem was to decide how to manage distance and isolated orders, waiting for other near them to appear (risking delays) or send semi-empty vans to delivery.

3.1 Theorical definition of the problem

The main features present in our case make it a *dynamic* and *stochastic* problem: the time of the simulation is divided in one day periods, every day some decisions are made keeping memory of the past (dynamic aspect), customers can refuse a delivery date proposed with a certain probability (stochasticity).

Vehicles are characterized by *capacity constraints*, in chapter **??** simulations outputs will be presented but will be limited to a *single vehicle case*. All tours start from a *single depot* with constraints on the *maximum duration* of each tour. At the end of the day vehicle as to comeback to depot. When customers place orders, they are promised delivery by a *due date*. Some orders concerns goods that are not immediately available and therefore become deliverable only after a *release date*.

3.2 Problem structure

The following algorithm better explain how orders are managed, from the receipt to the delivery. As well it highlights all the change in order status that can occur. The algorithm adopts a logic similar to those from use cases; this allow us to visualize how the problem is articulated, on the other hand it provide an overview on how it has been modelled in the developed tool.

Daily operations:

1. **Check new incoming orders.**

    (a) If there are new orders, the IT system check if ordered good is available

in inventory:

i. if product is available the order status is set “ready-to-delivery”;

ii. ii. otherwise, create an order to supplier. Set order status as “waiting-from-supplier” and set a release date.

(b) Otherwise, go to 2.

2. **Check the status of all orders and eventually update it, for each order:**

   (a) if occur the release date (previous status “waiting-from-supplier”), new order status “ready-for-delivery”;

   (b) if occur the due date (previous status “ready-for-delivery”), new status “beyond-the-deadline”;

   (c) if occur the shipment date (previous status “scheduled”), new order status “delivered”;

   (d) if occur the shipment date (previous status “scheduled-beyond-the-deadline”), new status “delivered beyond deadline”.

3. **Generate a list with only deliverables orders.**

4. **Apply a choice function to select a cluster of orders to be scheduled for next day delivery.**

   (a) If all customers accept the delivery date, go to 5;

   (b) if even just a customer refuse, exclude it from the list of deliverables orders, return to 4.

5. **Generate an optimized tour for the cluster of orders and set the shipping date for each, then:**

   (a) if previous status was "ready-for-delivery", change the status in "scheduled"

   (b) if previous status was "beyond-the-deadline", change the status in "scheduled-beyond-the-deadline";

6. **STOP, go to next day.**

Orders can have the following statuses:

- *"ready-for-delivery"*, is an order that can be scheduled on delivery
- *"waiting-from-supplier"*, an order characterized by a release date. The product was not available and has been ordered by a supplier. On the release date it is assumed that the asset is in the depot.
- *"beyond-the-deadline"*, when an order is on hold until it exceed the due date.
- *"scheduled"*, when an order is part of an already planned delivery tour.
- *"scheduled-beyond-the-deadline"*, as above but the order is delayed.
- *"delivered"*, after the delivery take place. The order is archived.
- *"delivered beyond deadline"*, as above but the delivery happen over the due date.

Every period the tool run a "daily routine". We assume that a certain number of orders arrive every day (1), some of these concern products already in stock, others concern customized goods that need to be ordered by a supplier. This assumption is quite realistic for a small furniture shop, as well as it is common for bulky or low demand products specially when they are not managed according to an inventory policy. After an update of the status of all orders (2) on hold is possible to move those that can be delivered (so "ready-for-delivery", "beyond-the-deadline" orders) to a separate list (3). The choice function (4) is the key element to achieve a good solution, in this case it returns a cluster of customers which involves a benefit in terms of duration (or distance) to serve in a single tour.

Chapter 4 is totally dedicated to the explanation of the function developed for the tool, the idea is that if the cluster is broken you need to search for another in order to have a good solution. The stochastic customer behaviour consists of the possibility to refuse a delivery on a certain date. If this happen, according to reasoning of cluster, a good solution is broken and the algorithm will iterate searching for another. When a solution is finally found and confirmed, it is possible to solve a simple Travelling Salesman Problem (TSP) obtaining for each vehicle a delivery tour (5).

3.3 Problem assumption

To model this complex problem a series of assumptions had to be added, some of these are strong and involve in a growing distance between the model and the reality but are necessary to avoid overburdening the simulation.

1. **Inventory management** is not part of the model.

   When a new order arrives a random number is generated and it is compared with a fixed probability, if it gets over the goods are assumed to be already in stock. The probability can be chose, the smaller it is the less likely new orders require a request to supplier (waiting-from-supplier status).

2. **Geographical dimension.**

   Depot is based in Turin (Italy). In order to address a of long-distance routing problem, customer can appear in seven contiguous region of North Italy: Valle d'Aosta, Piemonte, Liguria, Lombardia, Veneto, Trentino-Alto Adige, Friuli-Venezia Giulia, Emilia Romagna. We assume that this area is a realistic marketplace for the furniture shop; some location can be more than 500 km away, actually making a problem to serve some customer. In order to manage a specific number of possible customer locations we adopt a similar approach as Restrepo et al. (2019), where customers were aggregate depending on the geographic zone they belong using corresponding postal code. In this work we aggregate areas using the Italian system of municipality code (data provided by Istat). A database has been created containing 4423 possible location where a customer can appear, each one correspond to a pair of longitude, latitude coordinates. The original dataset is presented in Appendix. This assumption avoid to randomly generate coordinates which involves the risk that some are not accessible from the road network. Moreover in Bent, Hentenryck (2004) is shown as a geographical decomposition is a good approach to solve large-scale VRPs.

3. **Orders and location generation.**

   (a) *Customer locatio*n is randomly extracted by the locations database described above. To make the generation more realistic, the probability of a location being chosen randomly is proportional to the number of residents in that municipality.

   (b) *The volume of orders* is randomly drawn according to a probability distribution. The volume is modelled as multiples of a standard unit, can be considered a-dimensional.

   (c) *The release date* for new orders with the status "waiting-from-supplier" is randomly extracted from a probability distribution.

   (d) *Due date is deterministic*, but is possible to make it stochastic.

4. **Vehicles**

   (a) *Fleet size.* By default only one vehicle is set and the simulations set out in chapter 5 are all single-vehicle cases. This decision allows us to better analyse the functioning of the developed tool reducing the complexity of the outputs. However a multi-vehicle option (up to 3) is present in the tool, but actually it is not optimized properly.

   (b) Vehicle can work within a *day's working hours* set wide enough to allow the resource to reach even the most distant locations, avoiding infeasible delivery. A *service time* option is present in the code but not used in chapter 5 simulations in order to reduce the experimental parameters.

5. **Time horizon and time management**

   The horizon of simulation is split in period of one day. The scheduling of deliveries always take place for the next simulation day, a multi period scheduling can be a future extension of this work. The length of simulation is set at 168 days, this number is the result of a trade-off between a sufficiently long simulation period and an acceptable run time.

6. **Stochastic customer behaviour** The stochastic element of this work is the chance that any customer has to refuse a proposed delivery date. A strong assumption has been introduced around this, each customer can refuse a single time.

3.4 Simulation tool

The tool developed to solve this study case is programmed in python and uses object oriented programming (OOP) to better manage the order as a standard entity.

Each order created is characterized by the same set of attributes, these are designed to keep track of the order history during the simulation period. This approach follows a logic flexibility and scalability of the code. Some attributes are dynamically computed during the generation following the assumption discussed before, others derive from the database location selection (DB).

The main attributes of an order are:

- *customer ID*, assigned progressively;

- *Istat code* (DB);
- *municipality name* (DB);
- *distance from depot*, see next paragraph;
- *travel duration from depot*, see next paragraph;
- *longitude* (DB);
- *latitude* (DB);
- *volume*, randomly extracted;
- *order date*, corresponds to the generation date;
- *due date*, randomly extracted;
- *order status*, change dynamically during simulation;
- *shipment date*, added after the delivery is scheduled;
- *release date* (eventually) ;
- *route from depot information*, see next paragraph.

In order to be flexible with the depot selection (giving way to use the tool also for location routing problems), distance, duration are computed when the order is created moreover the road used from depot to the coordinates of the order is stored. A back-end computation of the shortest path (based on the real-road network) between two location was infeasible. As in Bruck et al. (2020) is used an open source routing

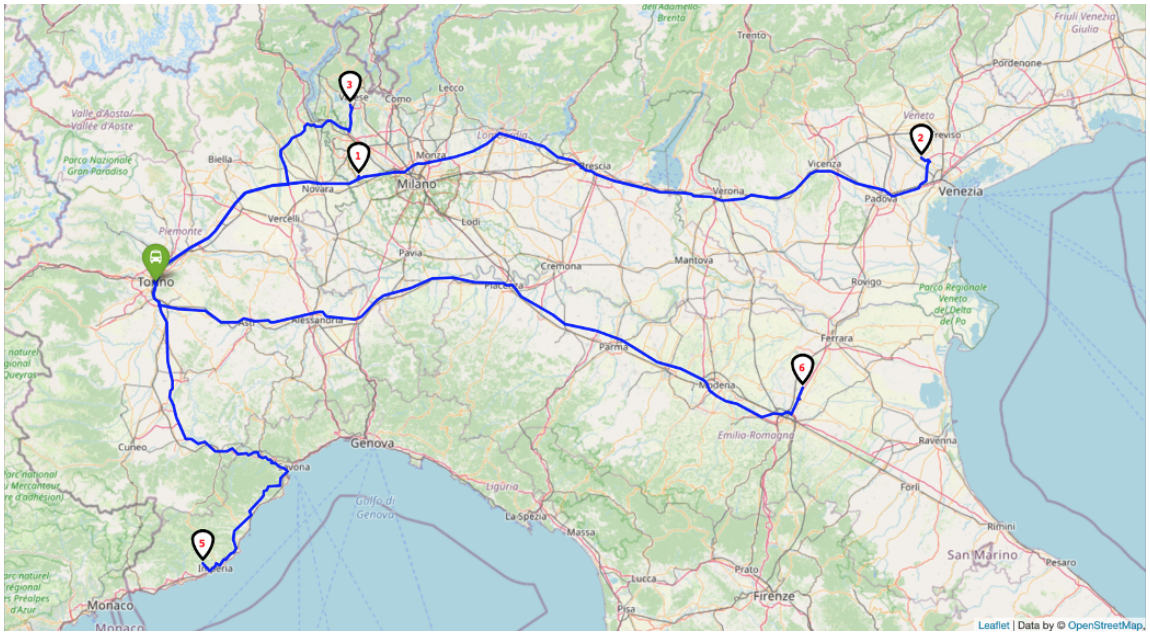

Figure 3.1: Random orders generated, plotted on maps

machine through real-time API requests; the service provider is OpenRouteService (ORS), the maps are powered by Open Street Map.

Given a pair of start and ending point the API returns a GeoJSON file, consisting in a collection of spatial geometries, analysing them it is possible to split the entire path into the individual sections of road travelled. Each section often correspond to a street, this allows us to compare the route from depot between different customer locations. Moreover on this information was studied the likelihood function used to select which order to delivery (chapter 4 is dedicated to explain the function) and solve the study case.

Figure 3.1 present the orders generated during a day of simulation, the shortest road are plotted on the map. This images came from a partial run of the tool.

Plotted orders are attributable to the following table (some information has been

cut out for a more comprehensive layout presentation). Here and in the continuation the unit of measure for distance are kilometres, minutes for duration. RTD means "READY-TO-DELIVERY" and WFS "WAITING-FROM-SUPPLIER": these status define the possibility or not to ship an order immediately.

ID	Munic.	Dist.	Dur.	Vol.	Order date	Due date	Status	Release
1	Mesero	115	76	53	2021-01-10	2021-01-17	RTD	
2	Scorzè	401	235	58	2021-01-10	21-01-17	RTD	
3	Buguggiate	148	100	39	2021-01-10	21-01-17	RTD	
4	Varano Borghi	132	92	67	2021-01-10	21-01-18	WFS	2021-01-11
5	Dolcedo	213	143	31	2021-01-10	21-01-17	RTD	
6	Bentivoglio	348	207	40	2021-01-10	21-01-17	RTD	

Table 3.1: Random orders generated related to figure 3.1

The simulation follows the daily routine introduced in section 4.1. To manage orders during the simulation are used different lists, a python data structure definable as a vector, where an order can be stored according to its current status. The tool provides a visual output of the run ongoing giving a real-time daily description of the events and, at the end of the run, a summary of the main statistics collected. Moreover it is possible to create and store daily maps of tours scheduled.

Figure 3.2 proposed an output example of the feedback provided by the tool during each day of a simulation.

Figure 3.3 is the maps representation of the scheduled tour from the previous output.

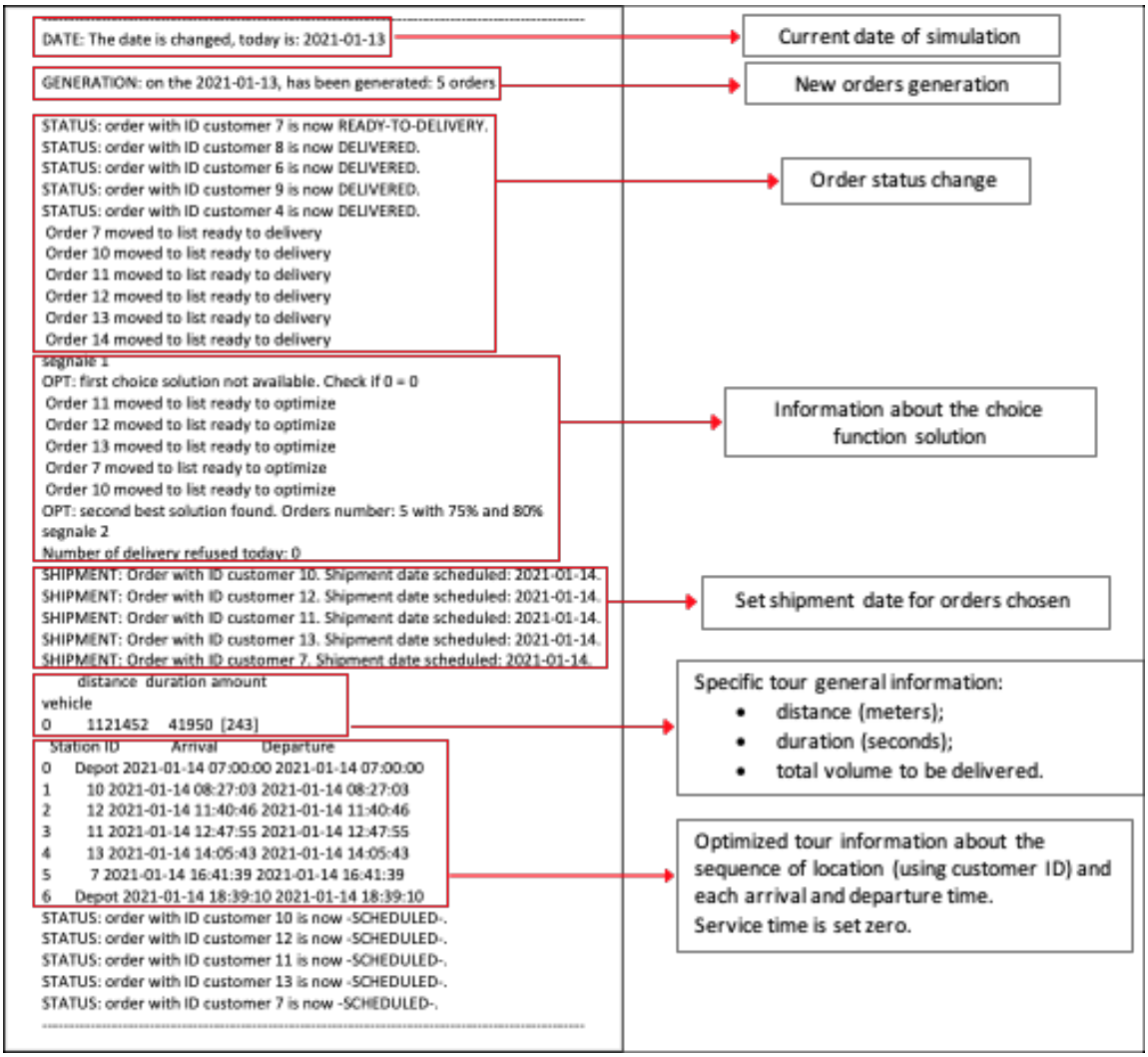

Figure 3.2: Daily output example

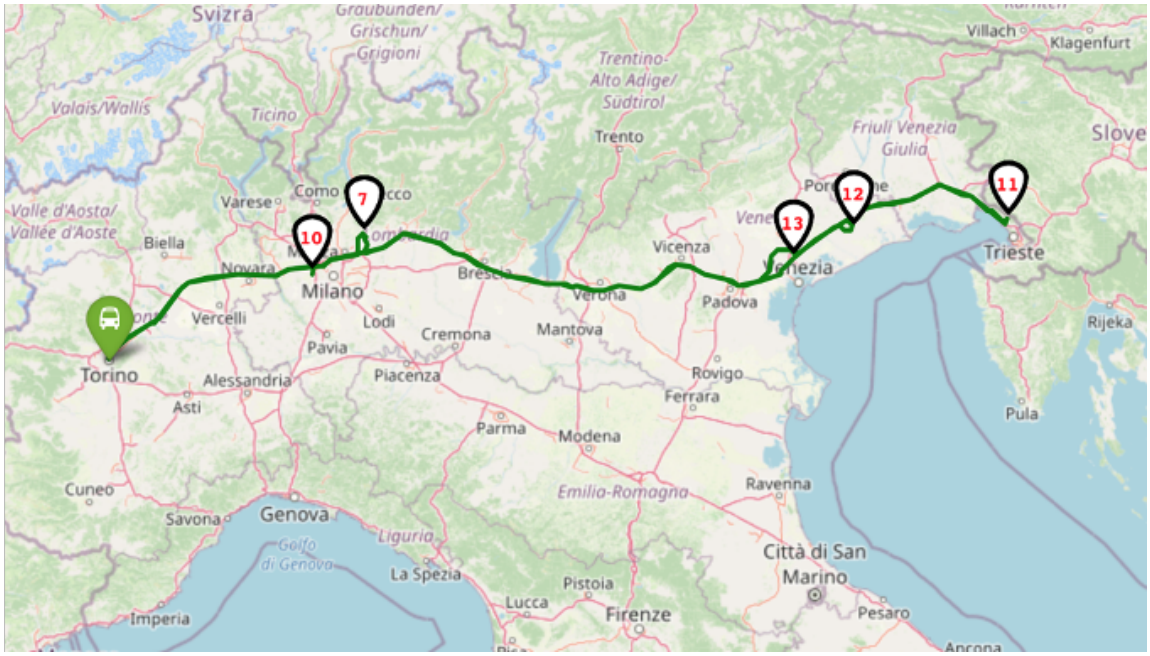

Figure 3.3: Optimized delivery tour for a cluster of locations

The tour creation is a simple TSP solution. It is computed, as for distance and duration, through an API request to OpenRouteService, then analysed to be printable on the map.

The example is taken from a run conducted with the aim of showing how the tool work, therefore the results of this run cannot be used to judge the goodness of optimization. The final summary of a run appears as in figure 3.4. It may change in future version of this tool.

The theorical contribution of this work is the study of the likelihood function applied during simulation to choice a good cluster of orders to schedule the next day, next chapter is dedicated to a deep analysis of how has been developed.

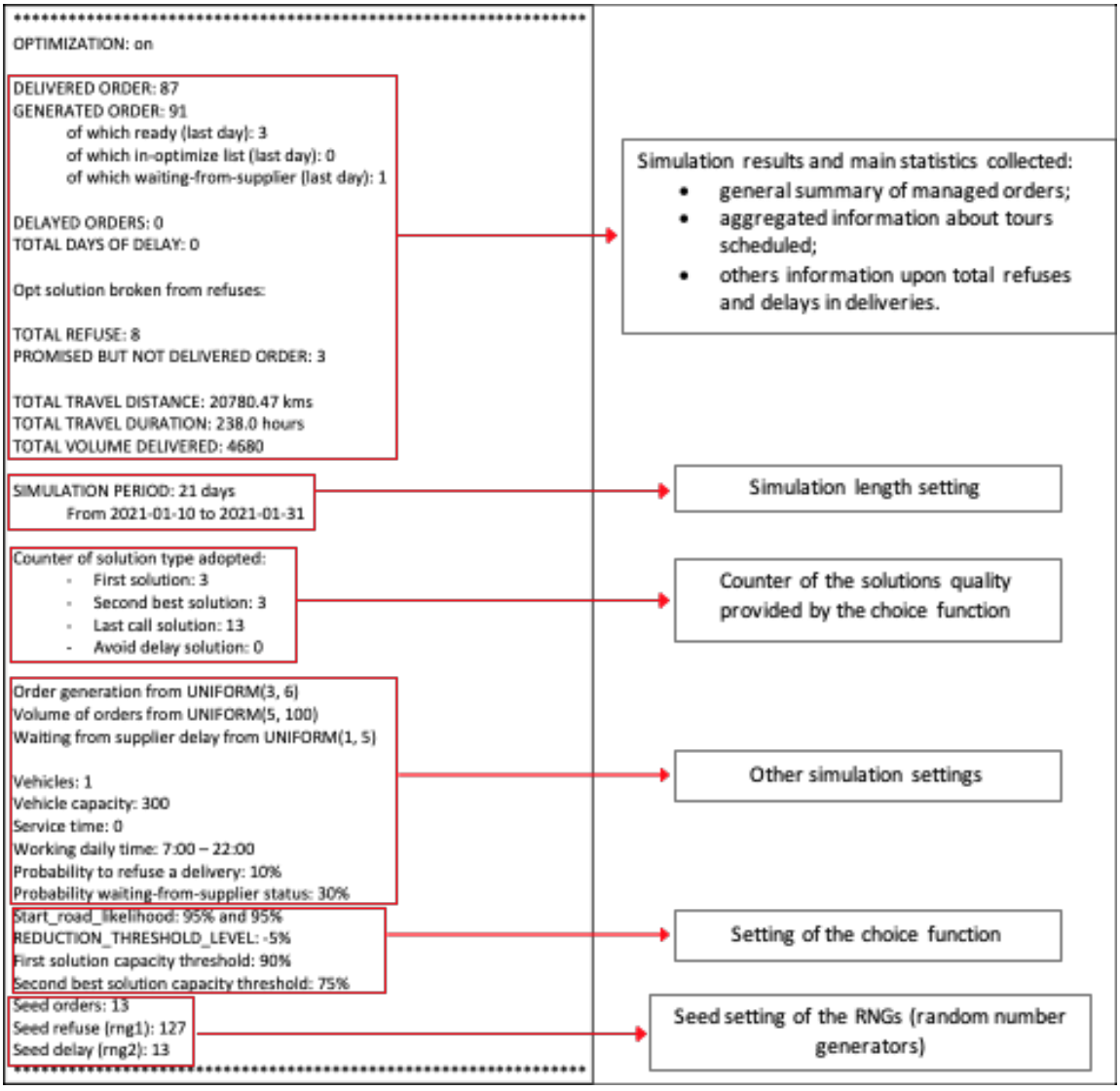

Figure 3.4: Final output example.

Chapter 4

Likelihood function

This chapter provide a description of the choice function developed ad-hoc for a large-scale routing scenario.

Firstly, the specific problem is defined, then an example of implementation contextualizes it in the daily routine and explains the logic behind his utilization. Finally, the function is exposed algorithmically, and it is shown how it has been implemented in the code.

4.1 Problem definition

Last chapter introduced the tool but left open his main mechanism, the function used to choose which order to schedule next day. The likelihood function is an iterative algorithm that can be applied to a list of orders. The objective is to identify a cluster of similar orders whose joint delivery can bring a benefit in term of cost; in this case total distance and duration are used as a cost proxy. Let's proceed step by step,

starting to illustrate how similarity between orders is computed.

In current literature a common approach to solve similar dynamic problem is to adopt a waiting strategy, setting some constraints that if verified open to a solution or, if not, force us to other choices. Literature approach is often based on a graph-based model. A theorical simulation based on the graph theory with fixed arcs bolded, simulating a real road network, is computationally feasible only for a small size problem. If we want to create a mathematical model, sized as this simulation, we need to manage a set of millions of fixed arcs each one characterized by different travel time (according to the specific road speed limitation). Then to compute a single travel we should execute hundreds of calculations, making it impossible to solve the problem in a short time.

In this work we want to decide the best moment to schedule a far order, and the solution obviously involves to keep waiting some of them. However, an order is not problematic if it is only distant, but when it is isolated too. To define when an order is isolated, we can choose one and observe its surroundings within a certain radius. If we contextualize this state with the road network, we can also say that an order is isolated when to reach it we do not pass by other orders.

In other words, even the most distant order is not isolated, then problematic, when we can make other deliveries along the same road and making small detours.

The algorithm proposed in this work rely on this concept, and exploits it building a likelihood matrix to compare all the orders and choose the more similar.

It is important to note that the road network includes highways, freeways, provincial and municipal road. To make them different is the speed limitation, therefore a longer distance does not necessarily involve a longer travel time.

This work also takes this aspect into consideration using the duration and not the distance for comparison between different paths.

The likelihood between two orders is computed comparing the "route from depot" data information and expressed in percentage terms. Chosen a pair of orders their roads are compared by their smaller segments, street by street, and the similarity end when the path divides. Then the time taken up to the division point is compared to the total duration of both paths.

Here are some examples obtained from program runs during which only two orders were generated; in this partial run the maps output report the istat code of the municipality from where the order come and a generic ID is assigned for a better comprehension of the percentage. It is possible to see the similar path plotted on a real map and read the corresponding similarity percentage to reach one location or the other.

Example 1 - figure 4.1 - table 4.1

From	ID	To	Duration [sec]	Distance [km]
Torino	1	Scorzè	5669	116
Torino	2	Gavazzana	5702	138

Table 4.1: Random orders generated related to figure 4.1

Common distance: 86,607 kms. Common duration: 3525,6 seconds.

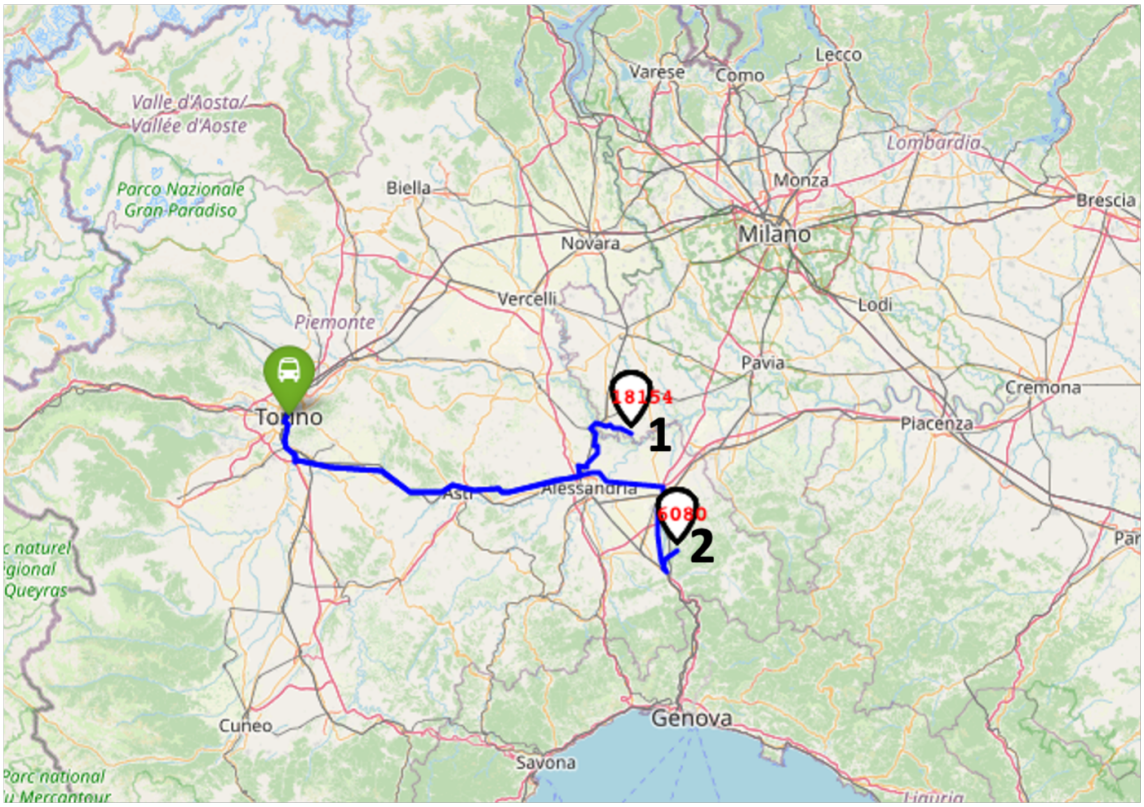

Figure 4.1: Pair of orders randomly generated, case 1.

Distance similarity:

- 74,29% (1 to 2);
- 62,66% (2 to 1).

Duration similarity:

- 62,19% (1 to 2),
- 61,83% (2 to 1).

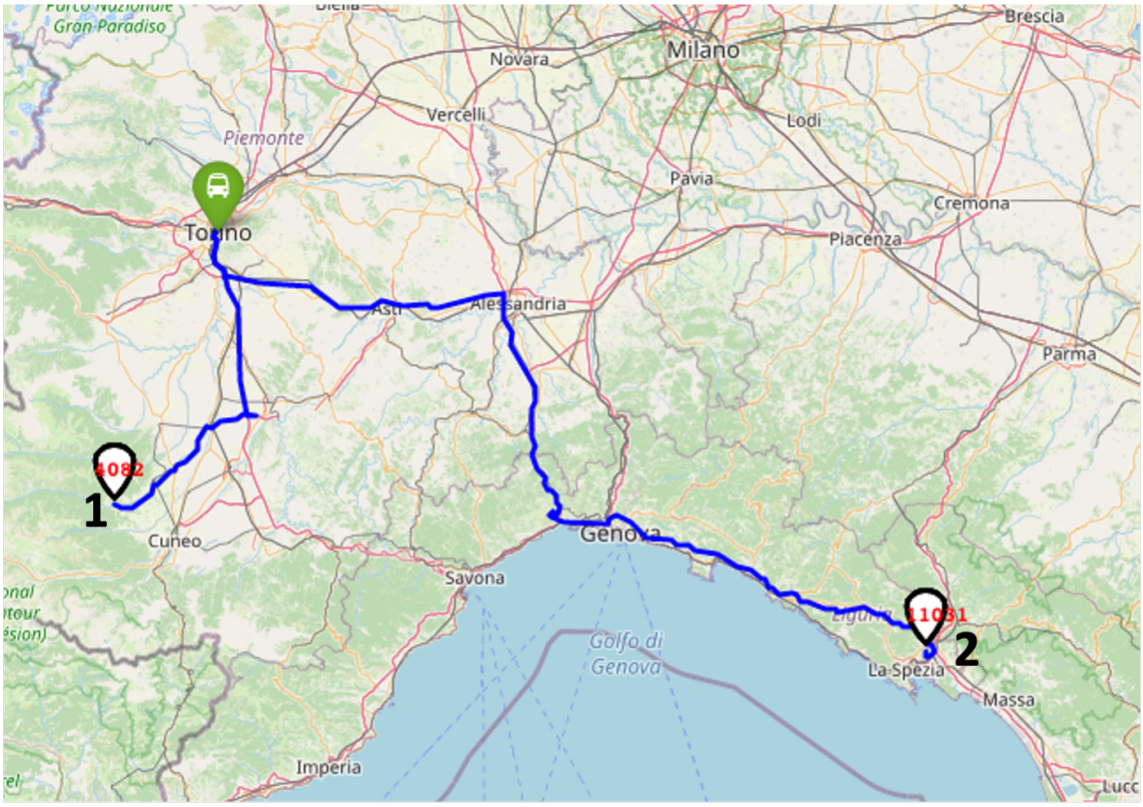

Figure 4.2: Pair of orders randomly generated, case 2.

Example 2 - figure 4.2 - table 4.2

Common distance: 13,819 kms. Common duration: 1112,4 seconds.

Distance similarity:

- 13,71% (1 to 2);
- 5,81% (2 to 1).

Duration similarity:

- 20% (1 to 2),

From	ID	To	Duration [sec]	Distance [km]
Torino	1	Dronero	5561	101
Torino	2	Vezzano Ligure	11076	267

Table 4.2: Random orders generated related to figure 4.2

- 10,04% (2 to 1).

Example 3 - figure 4.3 - table 4.3

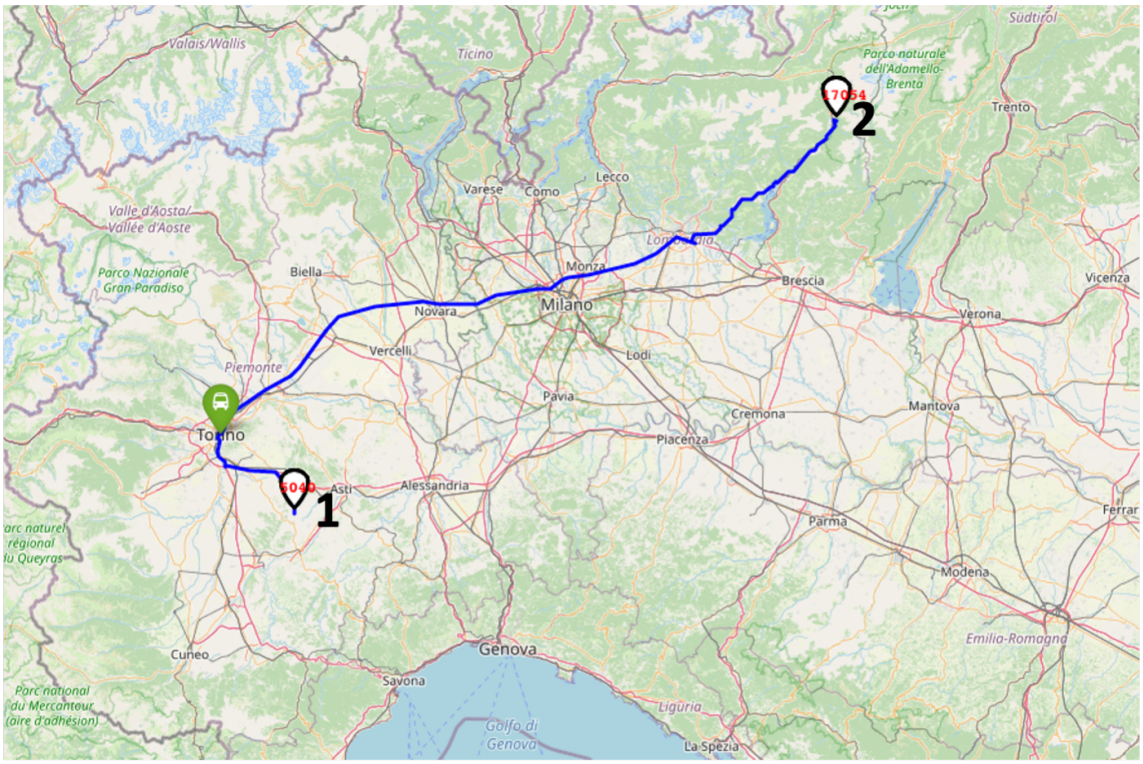

Figure 4.3: Pair of orders randomly generated, case 3.

Common distance: 0,34 kms. Common duration: 1112,4 seconds.

Distance similarity:

- 0,66% (1 to 2);

From	ID	To	Duration [sec]	Distance [km]
Torino	1	Cisterna d'Asti	3101	51
Torino	2	Cimbergo	12046	262

Table 4.3: Random orders generated related to figure 4.3

- 0,16% (2 to 1).

Duration similarity:

- 2,68% (1 to 2),
- 0,69% (2 to 1).

Analyzing these examples is possible to understand what happen if were used distances instead of durations for the comparison: the output information would be affected by a bias.

In the first the maps provide an explanation for the difference between similarity sets. The remaining path to the second location looks bigger but probably part of that road is a highway with higher speed limitation and therefore minor travel time. Because of this difference in the type of streets traveled the remaining duration is the same.

Something similar emerge from the second example. The common path is traveled inside a city, in kilometers the impact of speed limitation is not significant but in term of duration they are, the percentage double. The last example allows us to view two totally different paths.

This likelihood comparison is applied to pairs of orders coming from a list and returns a likelihood matrix. In the next section is proposed a scenario with multiple orders to show how it works.

4.2 Likelihood matrix application

4.2.1 Simple scenario (only volume and similarity constraints)

From a program run is obtained this demand scenario for a random day. The map below provides a graphic representation of the orders; in blue are highlighted the *"travel from depot"* for each order.

The depot is set in Turin. Durations are computed in minutes while distances in Kms. RTD means "ready-to-delivery", WFS means "waiting-from-supplier".

ID	Munic.	Dist.	Dur.	Vol.	Order date	Due date	Status	Release
1	Istriana	416	254	85	2020-12-10	2020-12-17	RTD	-
2	Megliadino S.V.	381	225	95	2020-12-10	2020-12-17	RTD	-
3	Benna	76	60	30	2020-12-10	2020-12-17	RTD	-
4	Palanzano	306	218	45	2020-12-10	2020-12-17	RTD	-
5	Dovadola	423	258	35	2020-12-10	2020-12-17	RTD	-
6	Berra	400	269	60	2020-12-10	2020-12-17	RTD	-

Table 4.4: Random orders generated related to figure 4.4

This case presents a few orders, but all of them located in other regions and with significant travel times to reach.

Our main problem is deciding the tour composition for the next day without knowing where will appear new orders in the following day. For example, a (wrong) human choice could be to schedule 4 and 5 because the 5 looks distant; tomorrow or further a new order can appear close to 5, forcing us to make more complex choice in the future.

The key data to make the best choice are:

- *Distance/duration*, farther a location is, the more complex is to reach it. If

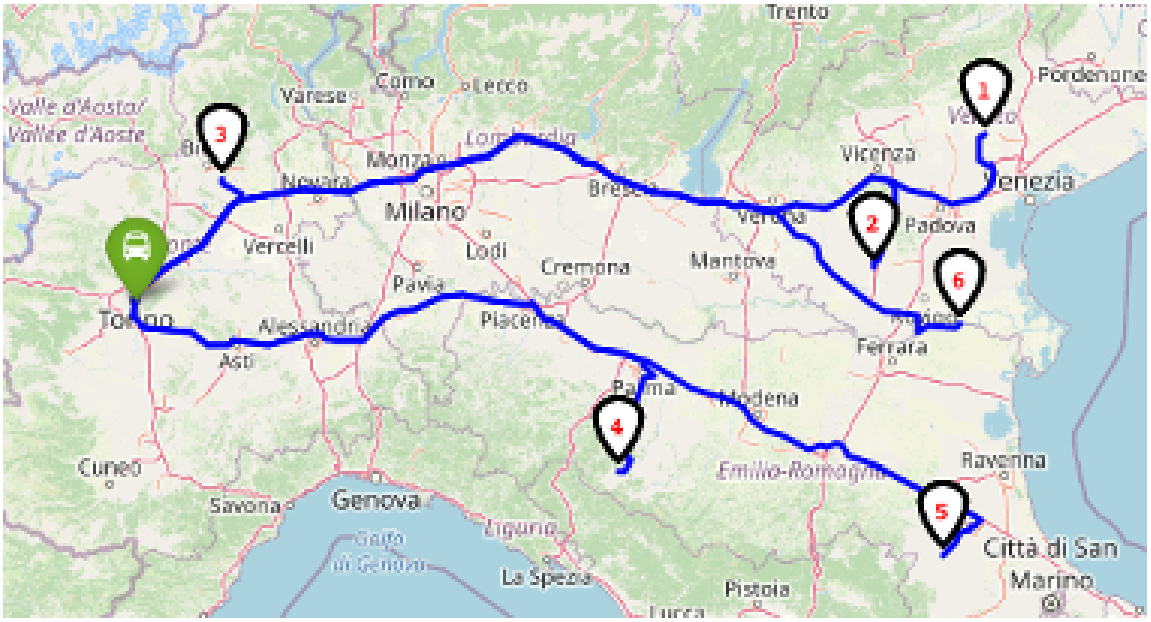

Figure 4.4: Randomly generated orders for the simple scenario.

we leave it pending, we can be forced to schedule an ad-hoc delivery to avoid delay;

- *due date*, the closer we are to the deadline, the fewer opportunities to schedule it in an optimal way are;
- *volume*, according to it we can fill the vehicle optimally.

The road likelihood matrix is generated with the duration similarities data; it would appear as follow.

From this matrix we can assert that the most similar orders are 1 and 2, respectively the first shares the 78,02% of the path with the second and observing in the opposite direction the similarity grow to 88,21%. The same reasoning for all the pairs.

Likelihood [%]	1	2	3	4	5	6
1	-	78,02	16,63	1	1	66,64
2	88,21	-	18,8	0,62	0,62	75,35
3	70,34	70,34	-	2,31	2,32	70,34
4	1,17	0,64	0,64	-	65,43	0,73
5	0,99	0,54	0,54	55,26	-	0,62
6	63,01	63,01	15,72	0,59	0,59	-

Table 4.5: Likelihood matrix. Simple scenario.

The matrix is used to identify a cluster of orders which can be reached along the same road. In order to identity this cluster we need to introduce threshold that suggest us when a solution is good or not.

- The **first threshold** regards the minimum percentage of similarity. We can choose a starting level and then go to reduce it until a solution is found.
- The **second threshold** is instead a constraint that allows us to accept a solution, seen the problem structure it is a minimum vehicle saturation level.

In the current example we can start whit a similarity threshold of 90% and 90% and a minimum vehicle saturation of 80% to a vehicle whit 300 as capacity (240). By iterating reducing the first threshold, the first feasible solution is obtained whit 78% and 78%, it consists in the cluster 1, 2. The total volume to delivery would be however below the minimal level (180 ¡ 240) so we continue to iterate. With 63% and 63% of similarity we obtain a new solution, the cluster 1, 2, 6. The cumulated volume is 240 and the solution is accepted. Then we solve a simple TSP (because we have only one vehicle) and schedule a tour for these customers.

This procedure stops at the first solution, which can potentially lead to not

considering better solutions. This is a specific choice. The reason is the sequence of pair compared, in this example the tool compared following the list orders, so: (1,2), (1,2), (1,3), (1,4), (1,5), (1,6), (2,1) etc. When the comparison meets the threshold, we store the orders and go on until the capacity level is reached.

Before starting the procedure, we should sort the list in a smart manner.

This example is simplified to provide a first look at the algorithm. There are two main elements that have not been considered: what to dealt with orders close to expiration, what to do if a cluster customer does not accept delivery.

4.2.2 Complex scenario (additional constrain on due date and stochasticity)

To give priority to expiring orders before the application of the algorithm we have to sort the list according to the due date. Then a second sorting can be applied based on the maximum duration to reach a location. In this way the first order to be considered is always the most problematic, the farthest and the next to expire. To provide a better explanation we can modify the previous example adding additional orders and varying some due date.

ID	Munic.	Dist.	Dur.	Vol.	Order date	Due date	Status	Release
1	Istriana	416	254	85	2020-12-15	2020-12-21	RTD	-
2	Megliadino S.V.	381	225	95	2020-12-11	2020-12-18	RTD	-
3	Benna	76	60	30	2020-12-12	2020-12-19	RTD	-
4	Palanzano	306	218	45	2020-12-13	2020-12-20	RTD	-
5	Dovadola	423	258	35	2020-12-11	2020-12-18	RTD	-
6	Berra	400	269	60	2020-12-10	2020-12-17	RTD	-
7	Arsiero	389	235	30	2020-12-11	2020-12-18	RTD	-
8	Breno S.V.	251	191	40	2020-12-13	2020-12-20	RTD	-
9	Ponte dell'Olio	204	139	40	2020-12-10	2020-12-17	RTD	-
10	Tornaco	115	83	75	2020-12-11	2020-12-19	RTD	-
11	Dolcedo	213	143	30	2020-12-11	2020-12-19	RTD	-
12	Brusasco	45	43	90	2020-12-10	2020-12-20	WFS	2020-12-13

Table 4.6: Complex scenario generated orders. Related to figure 4.5

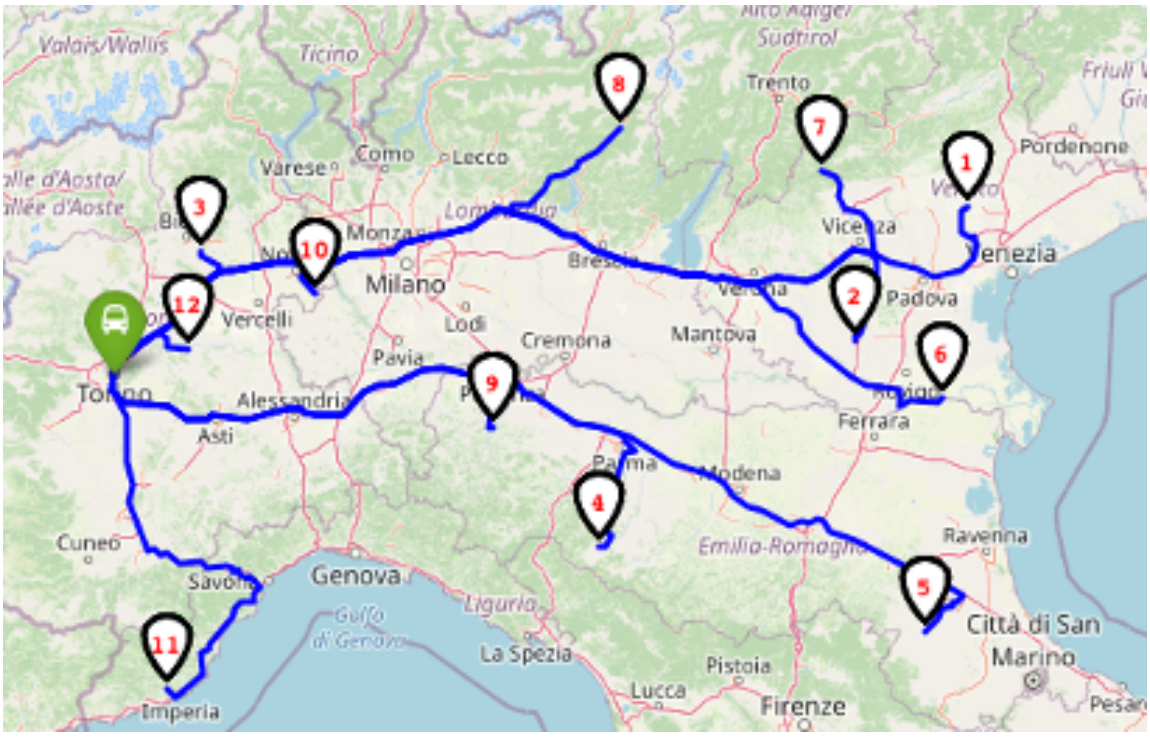

Figure 4.5: Randomly generated orders for a complex scenario.

Orders 12 is not ready to be delivered, so we can cut it from the list. Then apply: 1. EDD (earliest due date) sorting; 2. maximum duration sorting, to orders with same due date.

1. EDD (earliest due date) sorting;

2. maximum duration sorting, to orders with same due date.

In the next section will be explained why this first sorting is needed. This sorting is applied at any step of any iteration, however it is implied in this example. Table 4.7 is the output of these sorting.

ID	**Munic.**	**Dist.**	**Dur.**	**Vol.**	**Order date**	**Due date**	**Status**
6	Berra	400	269	60	2020-12-10	2020-12-17	RTD
9	Ponte dell'Olio	204	139	40	2020-12-10	2020-12-17	RTD
5	Dovadola	423	258	35	2020-12-11	2020-12-18	RTD
7	Arsiero	389	235	30	2020-12-11	2020-12-18	RTD
2	Megliadino S.V.	381	225	95	2020-12-11	2020-12-18	RTD
11	Dolcedo	213	143	30	2020-12-11	2020-12-19	RTD
10	Tornaco	115	83	75	2020-12-11	2020-12-19	RTD
3	Benna	76	60	30	2020-12-12	2020-12-19	RTD
8	Breno S.V.	251	191	40	2020-12-13	2020-12-20	RTD
4	Palanzano	306	218	45	2020-12-13	2020-12-20	RTD
1	Istriana	416	254	85	2020-12-15	2020-12-21	RTD

Table 4.7: Complex scenario, sorted orders.

Now it is possible to start looking for a solution. The likelihood matrix follows.

Likelihood [%]	**6**	**9**	**5**	**7**	**2**	**11**	**10**	**3**	**8**	**4**	**1**
6	-	0,59	0,59	63,01	63,01	0,59	22,67	15,72	43,44	0,59	63,01
9	1,15	-	76,33	1	1	13,34	1	1	1	76,33	1,83
5	0,62	41,19	-	0,54	0,54	7,2	0,54	0,54	0,54	55,25	0,99
7	72,1	0,59	0,59	-	84,56	0,59	25,94	17,99	49,7	0,59	84,41
2	75,35	0,62	0,62	88,36	-	0,62	27,1	18,8	51,94	0,62	88,21
11	1,11	12,92	12,92	0,97	0,97	-	0,97	0,97	0,97	12,92	1,58
10	73,36	1,67	1,67	73,36	73,36	1,67	-	50,89	73,36	1,67	73,36
3	70,34	2,31	2,31	70,34	70,34	23,31	70,34	-	70,34	2,31	70,34
8	61,14	0,73	0,73	61,14	61,14	0,73	31,9	22,13	-	0,73	61,14
4	0,73	48,62	65,23	0,64	0,64	8,5	0,64	0,64	0,64	-	1,17
1	66,64	1	1	78,02	78,02	0,89	23,97	16,63	45,94	1	-

Table 4.8: Likelihood matrix. Complex scenario.

The algorithm fixes the first order and then start creating the matrix, if a feasible solution is found it stops. Compared to the previous case, an improvement has been made to the algorithm: the similarity threshold are not reduced together. The start from the same level (i.e. 90% and 90%) then the first is fixed while the second is reduced until it reaches 1%. This premise to hold the farthest order and search for close others, then to search orders that are along the same road therefore have a very similar road with a short detour (i.e. observe order 8 and 10). If any solution is found both the threshold are reduced and the procedure iterate again.

It is proposed a solution for this demand scenario. Set:

- starting similarity threshold (SST) = 80% and 80%;
- reduction threshold level (RTL) = -5%;
- vehicle capacity = 300;
- min capacity vehicle saturation = 90% (270).

This example returns a solution in few steps.

Iteration 1 (min capacity vehicle saturation = 90%)

1. **Step 1. SST = 80% and 80%** return:

   7-2 (84,55%) and 2-7 (88,36%).

   Only orders 2,7 are selected.

   $TotalVolume : V(2,7) = 95 + 30 = 125 \leq 270$

   (**CONTINUE** with 80% and 75%)

2. **Step 2. SST = 80% and 75%** return (the second threshold is reduced according to the RTL until it reaches 1%):

   7-2 (84,55%) and 2-7 (88,36%)

   7-1 (84,41%) and 1-7 (78,02%)

   2-1 (88,21%) and 1-2 (78,02%)

   Orders = 1,2,7

$TotalVolume : V(1,2,7) = 85 + 95 + 30 = 210 \leq 270$

(**CONTINUE** with 80% and 70%)

3. **Following steps [...]** do not return any feasible solution. The selected order remains the same as in step 2.

Iteration 2 (min capacity vehicle saturation = 90%)

1. **Step 1. SST = 70% and 70%** return:

   7-2 (84,55%) and 2-7 (88,36%)

   7-1 (84,41%) and 1-7 (78,02%)

   2-1 (88,21%) and 1-2 (78,02%)

   Orders = 1,2,7

   $Totalvolume : V(1,2,7) = 85 + 95 + 30 = 210 \leq 270$

   (**CONTINUE** with 70% and 65%)

2. **Step 2. SST = 70% and 65%** return:

   7-2 (84,55%) and 2-7 (88,36%)

   7-1 (84,41%) and 1-7 (78,02%)

   2-1 (88,21%) and 1-2 (78,02%)

   Orders = 1,2,7

   $Totalvolume : V(1,2,7) = 85 + 95 + 30 = 210 \leq 270$

   (**CONTINUE** with 70% and 60%)

3. **Step 3. SST = 70% and 60%** return:

   7-6 (72,1%) and 6-7 (63,01%)

   7-2 (84,55%) and 2-7 (88,36%)

   7-1 (84,41%) and 1-7 (78,02%)

   2-6 (75,35%) and 6-2 (63,01%)

   2-1 (88,21%) and 1-2 (78,02%)

   Orders = 1,2,6,7

   $Totalvolume : V(1,2,7) = 85 + 95 + 60 + 30 = 270 \leq 270$

   (STOP)

The cluster is defined but the next step is to call all the customers selected and propose them tomorrow as delivery date. If one of them refuse the entire cluster is compromised and a new iteration must be made.

If even reducing both the threshold to 1% and 1% any solution is returned the solution is to reduce the minimum capacity constraints and start again from the beginning.

In the tool has been added a constraint on the number of times a customer can refuse a delivery.

4.3 The solution algorithm

For the best of our knowledge this section provides the algorithm of the entire solution approach, including the first sorting to the cluster as output.

Given a list of orders we can identify, according to the problem definition, ready-to-delivery or waiting-from-supplier orders, only the first are deliverable. Then we define a list R containing these orders. The algorithm works as follow. The expected output is a shortlist called O, a cluster that we want to delivery; as well a value for Q, the total volume that will be delivered linked to the list O. We can set the thresholds and the RTL (reduction threshold level) to modify the behaviour of the algorithm. Instead, we can act on the list of orders L changing their features to simulate another scenario.

L = order list

O = list of selected orders

t_1 = similarity threshold level first comparison entity to second

t_2 = similarity threshold level second comparison entity to first

$R.T.L.$ = Reduction Threshold Level, is the reduction applied to t_1 and t_2 at any failed iteration

C_{min} = min. capacity level, it constrains to a minimum level of saturation of the vehicles

C_{max} = max. capacity level of the vehicle

t_{ij} = likelihood coefficient of order i compared to j

t_{ji} = likelihood coefficient of order j compared to i

Q = sum of volume of the selected orders (cumulative value, if an iteration fail it is reset to zero).

q_i = volume of the order i

q_j = volume of order j

Algorithm 1: Likelihood function algorithm

```
Data: list L of orders with i, j pointers for orders.
Result: list O of selected orders.
initialization;
set t_1, t_2, C_min, C_max, RTL, Q;
while t_1 ≥ 1 do
    while t_2 ≥ 1 do
        for i ∈ L do
            for j ∈ L and i ≠ j do
                if t_ij ≥ t_1 and t_ji ≥ t_2 then
                    Q = Q + q_i + q_j;
                    if Q ≥ C_min and Q ≤ C_max then
                        O = O + {i,j};
                        STOP, Return list O
                else
                    Q = Q − q_i − q_j;
                    j = j + 1;
            Q = 0;
            i = i + 1;
        t_2 = t_2 − RTL;
    t_2 = t_2,initial;
    t_1 = t_1 − RTL;
```

4.4 Likelihood function implementation

Respect to a theorical scenario in a real application we need to consider a few more constraints. For example, a careful reader may have already noticed a weakness in a "as is" implementation of the function: in low demand scenario, we have orders that remain isolated for long time. Some more consideration will explain how these aspects were taken into consideration.

4.4.1 Sorting every iteration

In the previous section I dealt with the likelihood function explaining the logic behind its choice. We can remember that the actual algorithm stops when it finds a solution, not the best but a feasible one. The problem rises thinking about the "problematic orders" that are distant and isolated, if we do not consider them for too long, they became even more complicated to manage. In fact, the algorithm does not take into consideration the due date as a constraint, so some orders may be late by exceeding the due date by a lot.

To avoid excessive delay has been added the sorting at the beginning of any algorithm iteration, we have to apply:

1. *EDD (earliest due date) sorting;*

2. *maximum duration sorting, to orders with same due date.*

In this way the first solutions searched always include the firsts orders of the lists,

that are the next to go beyond the deadline (if they are not already), and the most distant (bigger duration).

We can say that this is an "home-made" solution tailored for this implementation, future development of this algorithm should better consider the deadline aspect inside the main algorithm.

4.4.2 Avoid delay condition and specific solution

The sorting is a partial solution but it is not enough to avoid "no-sense" lateness during a run. If a problematic order is always the first so it is included in any solution search but never scheduled for a delivery, we could stop it for too long. To simplify we can imagine that for months no neigh-boring orders are generated. It would be reasonable to sacrifice a good solution to avoid a delay? In this work the answer is affirmative.

In this case is coded a specific module that adopt a "simple-solution". It simply takes order from the list in the same order they are found, until the capacity constraint allows. The trigger for this solution is when we have orders expiring "tomorrow". We will lose efficiency, but we avoid delays.

4.4.3 Low demand scenario

The simple solution explained above is used also in another situation. The algorithm creates a cluster using two thresholds, the first based on similar duration while the

second on vehicle capacity limit. What happen if we do not have enough order to fulfil the condition about the capacity, if the total volume is not enough, we should not delivery anything? Even if this answer should be the result of another trade-off in the present work we decide to delivery every time there is an order. So has been set a trigger that calls the simple solution if it is impossible to find a solution with the algorithm because of a low volume scenario.

Future formulation of this algorithm could bring to different way to address these implementation issues.

Chapter 5

Simulation output analysis

After the discussion on the specific problem (chapter 3), the solution approach (chapter 4), in the following pages will be shown different output of the simulation tool with the aim to identify how good is this approach and its weaknesses.

This part of the work wants to go into the technical details of the tool allowing to understand which parameters can be set and how. Furthermore, will be explained the configurations of the tool with which tests have been performed and then how an overall performance assessment was performed. Finally, the chapter will close with some considerations regarding the results of the tests performed, the weakness that emerged, and possible ideas for improvement for future analyses.

5.1 Benchmark

This work dealt with a hypothetical scenario based on the observation of real SMEs behaviour about long range delivery. This mean that there are any real data to use

as benchmark.

However, it is possible to create a benchmark according to the case scenario as it has been introduced in chapter 1. We described a furniture shop where deliveries where managed manually, so we can hypothesize that the strong growths demand has forced to reduce the time dedicated to scheduling deliveries.

The idea for a benchmark is to imitate this scheduling approach using an algorithm which manages deliveries by giving precedence to the next to go beyond the deadline. This is the simplest way to choose the next customer to serve, a sort of *"dumb-approach"*.

If we want to compare any simulation with a benchmark then a single simulation is indeed a pair of runs with the same set of customers in the same order (same seed for the random generation) but solved using different solution method: one with the likelihood function introduced in chapter 4, the other with a dumb solution strategy that schedule according to the due date order.

In other words, each simulation must be compared with its benchmark run where the likelihood algorithm it is off, and it is used instead a simple EDD logic and a vehicle capacity constraint to create clusters of orders to schedule.

The results presented in section 5.3 are the mean of the improvement/worsening from multiple couples of benchmark-optimized runs.

As mentioned above the seed used to set up the twin simulation (optimized and dumb) are the same, in this way we have the same demand scenario (customer, volume, etc. . .). In different pair of run the seed has been changed randomly.

5.2 Simulation parameters

A part of the tool is able to generate customers based on the setting given to a series of parameters. These elements regard the demand distribution, the probability to refuse an order etc... these will be our experimental parameters.

The simulation run have the purpose to evaluate the response of the algorithm according to different scenarios.

For reason of time, it was not possible to analyse in depth what was the best setting for the likelihood function, in these simulations it is set up according to what should be a good setting.

Firstly, it is important to introduce all the settings that can be modified. Some of them are related to the generation of customer, for example it is possible to choose the demand distribution both as regards the number of orders that can arrive each day and in term of their volume. In these simulations only uniform distributions have been used but nothing prevents from use normal, exponential or other statistical distribution.

Other parameters define the length of the simulation and other features more related to the delivery activity, as the service time spent at each customer location.

Other settings are related to the likelihood function. They can be changed at will, however at this stage the goal was to test the goodness of the algorithm when the demand scenarios changed, therefore only a specific configuration deemed sufficiently good was used.

In the future it will be possible to conduct in depth analyses to define which are

the optimal settings.

The generation parameters are the following:

- *days of simulation define the length of the simulation;*
- *likelihood function on/off, if turned off the run generate a benchmark;*
- *daily order distribution, (experimental);*
- *volume of order, it is set to vary according to a fixed probability distribution;*
- *waiting-from-supplier probability, (experimental) the stock-out probability;*
- *waiting-from-supplier waiting time, it is the time needed to get supplies from the supplier;*
- *due date delay, respect to the moment of generation of an order is the time when the due date is set;*
- *probability to refuse a delivery date(experimental);*
- *number of vehicles, the size of the fleet;*
- *vehicle capacity, (experimental) is the volume loadable on a truck;*
- *service time, it is the time spent to accomplish a delivery so to unload the orders and deliver it;*
- *start hour of a working day, the time from which vehicles can star delivering;*

- *ending hour of a working day, the time by which all vehicles must be at the depot.*

In table 5.1 is presented an example of setting for a pair of runs, the benchmark one and the optimized. In yellow the experimental parameters.

Days	Optimization	Daily ord	Volume	Waiting prob.	Waiting delay	Due date delay	Refuse prob.	Vehicle number	Vehicle cap.	Service time	Start work hour	End work hour
168	benchmark	Unif(3,6)	Unif(5,100,1)	10%	Unif(1,5)	7	10%	1	300	0	07:00	22:00
168	on	Unif(3,6)	Unif(5,100,1)	10%	Unif(1,5)	7	10%	1	300	0	07:00	22:00

Figure 5.1: Example of simulation parameters.

The experimental parameters will vary during simulations while the others are the chosen settings that are fixed across all the tests run.

Each simulation is long 168 days, 24 weeks. Each order has a random volume extracted from a uniform distribution with a minimum size of 5 and a maximum of 100; the step in the extraction is set at one, so no float but only integers number. The volume is a fictitious number with no unit of measure.

The due date delay is the time interval between when the order is generated and when the deadline is set. During the run used to evaluate the overall performance this parameter is set at 7 deterministic days. In other word when an order is generated the deadline to delivery is always set seven days later.

The vehicle number is set as one. The tool support at most 3 vehicles but it is not optimized to work properly with this setting. The complexity is in the cluster creation inside the likelihood function.

The service time is zero. This choice to simplify the simulations. An idea could be to make it proportional to the volume of the specific order.

The working time is a very stringent constraint as it does not allow exceptions. Each delivery tour can start after 7 am but must be completed (the vehicle parked at the depot) by 10 pm. Such a large time window is also made obligatory by the large geographical region from which the orders can arrive (the entire North Italy); a lower window would have made it impossible a priori to deliver to certain locations.

The following are instead the setting of the likelihood function.

- Similarity threshold level of the first order to compare with the second;
- similarity threshold level of the second order to compare with the first;
- minimum capacity saturation level of a vehicle;
- second best minimum saturation level of a vehicle;
- RTL, reduction threshold level.

Similarity duration 1 to 2	condition	Similarity duration 2 to 1	Min. capacity level	Second best cap. level	RTL
95%	and	95%	90%	75%	-5%

Figure 5.2: Example of likelihood function settings.

These settings are fixed and refer to the likelihood function introduced in the previous chapter.

The two-similarity duration percentage are the threshold about the similarity in the duration to reach one location over another, they are the percentage used to create the likelihood matrix and create a cluster of orders.

These similarity levels are the initial ones that, as described in the chapter 4, are progressively reduced until a cluster of customers is identified.

The reduction of these threshold at each iteration that does not lead to a solution occurs according to the RTL. The smaller the Reduction Threshold Level is, the greater the number of iterations of the likelihood algorithm will be, increasing the computational weight of the run. However, if the RTL is too big it turns out to be a too permissive filter.

In the test simulations the similarity initial level is set at 95% and is reduced according to an RTL set at 5%. These should be a good compromise.

In chapter 4 was identify a second big threshold, tied to the minimum saturation of the vehicle to accept a cluster. In the tool are implemented two different level of solution: a first best that can be found when a first level of saturation is reached, a second solution that is search iterating from the beginning the entire algorithm but using a less restrictive capacity constraint. This option is a choice in the code implementation; in future development a more dynamic capacity threshold could be implemented.

The performance of a simulation can be evaluated using a series of statistics that the tool collects autonomously during its operation.

At the end of each simulation the tool presents a summary of the run which

contains the following information.

- Number of orders delivered;
- number of orders generated;
- number of orders delivered beyond the deadline;
- sum of days of delays;
- total simulation time;
- total distance travelled;
- total travel duration;
- total volume delivered.

Delivered	Generated order	Order delayed	Total days delayed	Simulation time [min]	Total Distance Traveled [km]	Total Travel Duration [h]	Total Volume Delivered
745	749	1	7	19	171.346	1.996	39.346
738	745	2	4	19	156.440	1.845	39.003

Figure 5.3: Example of collected statistics.

The number of orders generated depends on the distribution of daily orders and the number of simulation days. Two examples in the image above.

Given a total number of orders most of them are delivered, some beyond the deadline. Each order delivered after its due date increase the counter of "order delayed", and at the same time the entity of that delay is summed to the counter that track the total number of days of delay.

The simulation time is strongly dependent by the API used to compute distance and durations from depot when an order is generated. This API allows you to have detailed information on the route (indispensable for the creation of the likelihood matrix and the creation of clusters) but is limited in the number of calls per minute, up to 40.

It is important to say that using other provider, it could be possible to remove this limitation, however ORS turns out to be one of the most reliable to offer this resource for free. The main statistics used in the next sections are the total distance, total duration and total volume delivered. These information are collected during the simulation increasing these counters every time a tours is created.

In addition to this information, it is possible to collect further.

5.3 Simulation

We will analyse the results of four different scenario. As told before the algorithm setting will not change and the performance are evaluated according to total distance, duration and volume. We will see four main variants where each experimental parameter will be made to vary and the impact on the statistics will be evaluated.

5.3.1 Daily order distribution variation

The first parameter to be varied is related to the number of orders generable each day. The purpose is understanding how the tool perform with an increasing variability.

The demand distribution is a Uniform with the mean fixed at 4,5 orders per day.

The minimum and maximum values of the uniform change while maintaining the mean.

The daily order distribution scenario are the following:

- Uniform(0,9);
- Uniform(1,8);
- Uniform(2,7);
- Uniform(3,6).

The other experimental parameters are set as follow:

- waiting from supplier probability = 10
- refuse probability = 10
- vehicle capacity = 300.

The results of the simulations are summarized in the following tables and graphs.

Total Distance Traveled [km]

Daily Order Distribution	Benchmark	Optimized	Comparison
Unif(0,9)	168.297	160.892	-4,6%
Unif(1,8)	169.351	159.692	-6,0%
Unif(2,7)	173.081	162.976	-6,2%
Unif(3,6)	172.268	160.497	-7,3%

Total Volume Delivered

Daily Order Distribution	Benchmark	Optimized	Comparison
Unif(0,9)	1.953	1.881	-3,8%
Unif(1,8)	1.962	1.862	-5,4%
Unif(2,7)	2.008	1.904	-5,5%
Unif(3,6)	2.001	1.885	-6,2%

Total Volume Delivered

Daily order distribution	Benchmark	Optimized	Comparison
Unif(0,9)	38.740	38.476	-0,7%
Unif(1,8)	38.516	37.968	-1,4%
Unif(2,7)	39.267	38.979	-0,7%
Unif(3,6)	38.998	38.671	-0,8%

Daily Order Distribution	Distance perf.	Duration Perf.	Volume perf.
Unif(0,9)	-4,6%	-3,8%	-0,7%
Unif(1,8)	-6,0%	-5,4%	-1,4%
Unif(2,7)	-6,2%	-5,5%	-0,7%
Unif(3,6)	-7,3%	-6,2%	-0,8%

Figure 5.4: Simulation output with daily order distribution variation.

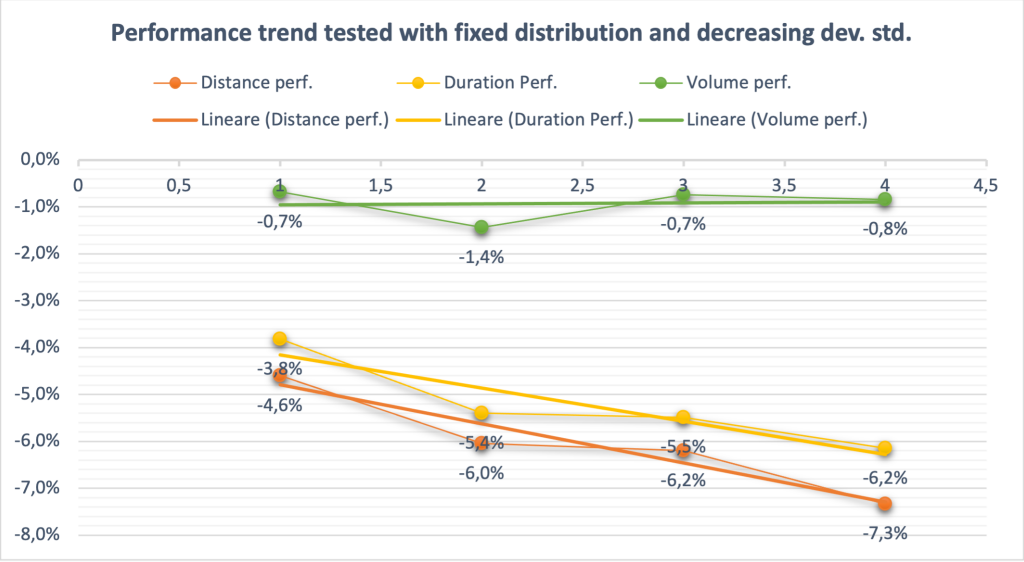

Figure 5.5: Daily order distribution variation, performance trends.

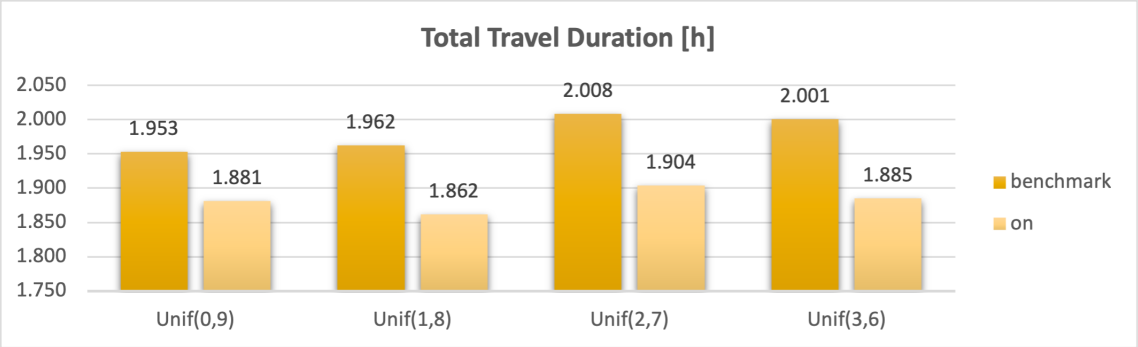

Figure 5.6: Daily order distribution variation, total travel duration reduction.

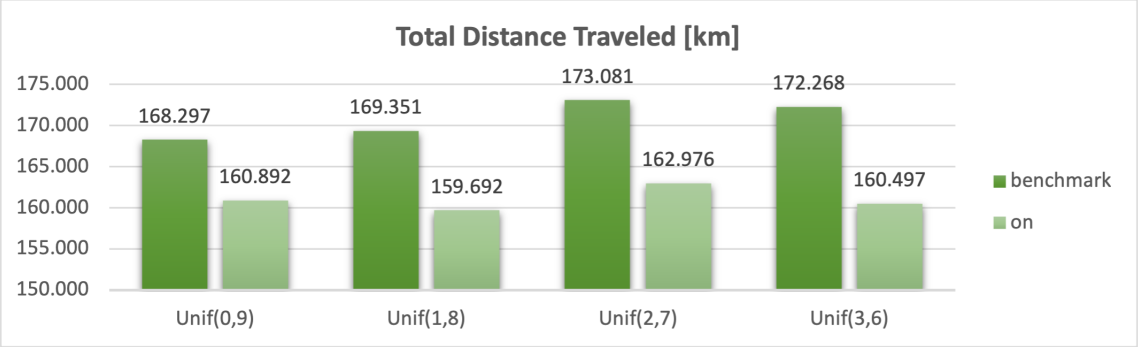

Figure 5.7: Daily order distribution variation, total distance traveled reduction.

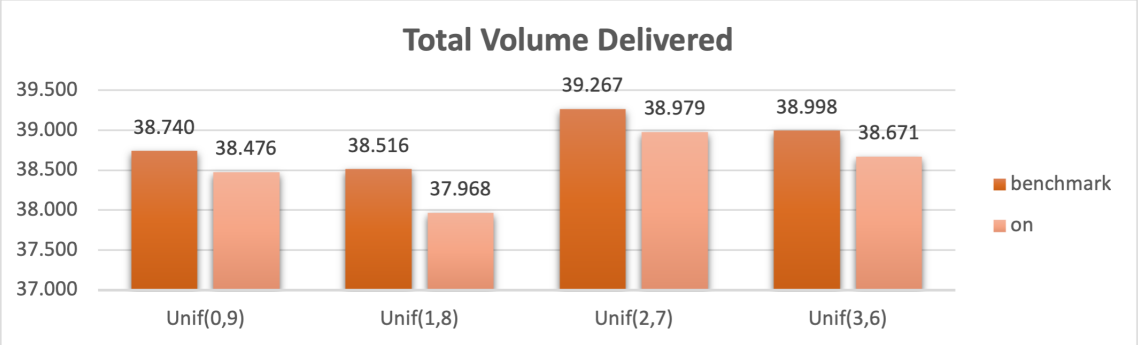

Figure 5.8: Daily order distribution variation, total volume delivered reduction.

The variability does not seem to have a significant impact on the volume deliv-

ered, in each scenario there is a minimal improvement over the benchmark. Total distance and duration are significant reduced by the adoption of the likelihood function algorithm however the improvement is reduced as the variability increases. Both distance and duration follow a similar worsening.

5.3.2 Waiting from supplier probability variation

The waiting probability represent the case when a customer asks for a product that is not available and must be ordered to a supplier.

In this case the compared scenario are the following:

- waiting prob. = 0%;
- waiting prob. = 10%;
- waiting prob. = 20%;
- waiting prob. = 30%.

The other experimental parameters are set as follow:

- waiting from supplier probability = 10%;
- refuse probability = 10%;
- vehicle capacity = 300.

The results of the simulations are summarized in the following tables and graphs.

Total Distance Traveled [km]

Waiting Probability	Benchmark	Optimized	Comparison
0%	170.117	157.060	-8,3%
10%	172.268	160.497	-7,3%
20%	172.783	161.539	-7,0%
30%	171.810	160.778	-6,9%

Total Volume Delivered

Vehicle Capacity	Benchmark	Optimized	Comparison
0%	1.964	1.832	-7,2%
10%	2.001	1.885	-6,2%
20%	1.992	1.886	-5,7%
30%	1.996	1.879	-6,2%

Total Volume Delivered

Waiting Probability	Benchmark	Optimized	Comparison
0%	38.569	38.304	-0,7%
10%	38.998	38.671	-0,8%
20%	38.493	38.532	0,1%
30%	38.951	38.687	-0,7%

Waiting Probability	Distance perf.	Duration Perf.	Volume perf.
0%	-8,3%	-7,2%	-0,7%
10%	-7,3%	-6,2%	-0,8%
20%	-7,0%	-5,7%	0,1%
30%	-6,9%	-6,2%	-0,7%

Figure 5.9: Simulation output with waiting from supplier probability variation.

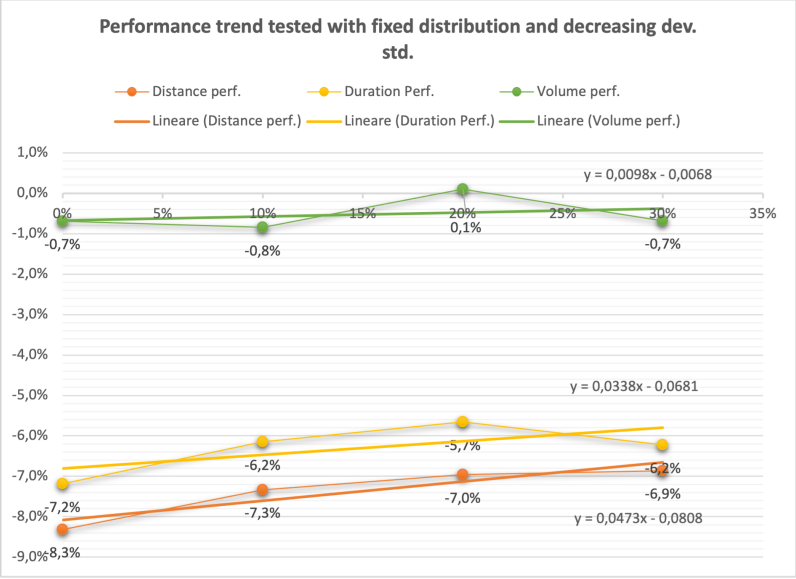

Figure 5.10: Waiting probability variation, performance trends.

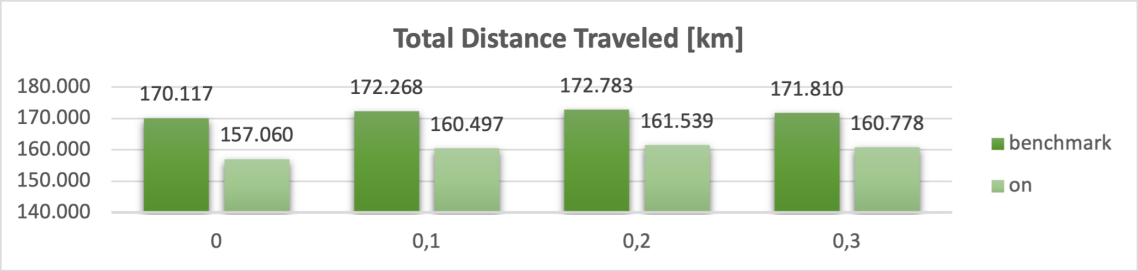

Figure 5.11: Waiting probability variation, total travel duration reduction.

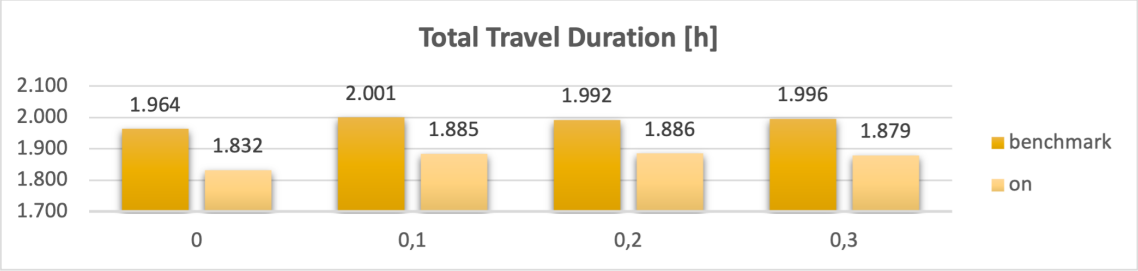

Figure 5.12: Waiting probability variation, total distance traveled reduction.

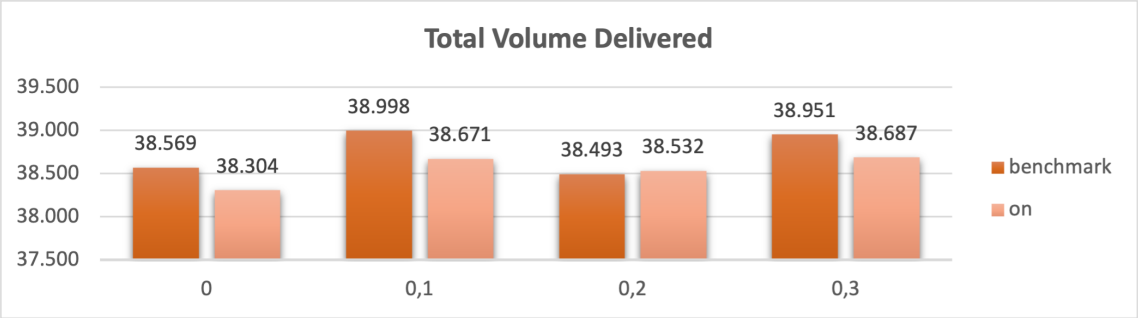

Figure 5.13: Waiting probability variation, total volume delivered reduction.

The waiting from supplier probability is a form of stochasticity. The increasing probability bring to a worsening in performance, the difference in performance compared to benchmarks progressively is decreased. As in the previous case both

distance and duration have a similar behaviour. Even the delivered volume looks similar to the first case, the increasing variability does not generate any significant trends in performance and the difference between an optimized and a benchmark scenario stay flat.

5.3.3 Probability to refuse a delivery variation

The next parameters that will be varied concern the opportunity of a customer to refuse a delivery date. The major it is more frequently a cluster selected from the likelihood function is broken. From a computational side this does not affect the duration of a simulation.

For these parameters will used the following settings:

- refuse prob. = 0
- refuse prob. = 10
- refuse prob. = 20

The other experimental parameters are set as follow:

- daily orders distribution = Uniform(3,6);
- waiting probability = 10
- vehicle capacity = 300.

The results of the simulations are summarized in the following tables and graphs.

Total Distance Traveled [km]			
Vehicle Capacity	Benchmark	Optimized	Comparison
0%	173.139	167.765	-3,2%
10%	172.268	160.497	-7,3%
20%	170.692	160.291	-6,5%

Total Volume Delivered			
Vehicle Capacity	Benchmark	Optimized	Comparison
0%	2.006	1.954	-2,6%
10%	2.001	1.885	-6,2%
20%	1.960	1.856	-5,6%

Total Volume Delivered			
Refuse Probability	Benchmark	Optimized	Comparison
0%	39.750	39.840	0,2%
10%	38.998	38.671	-0,8%
20%	39.207	38.696	-1,3%

Refuse Probability	Distance perf.	Duration Perf.	Volume perf.
0%	-3,2%	-2,6%	0,2%
10%	-7,3%	-6,2%	-0,8%
20%	-6,5%	-5,6%	-1,3%

Figure 5.14: Simulation output with probability to refuse a delivery variation.

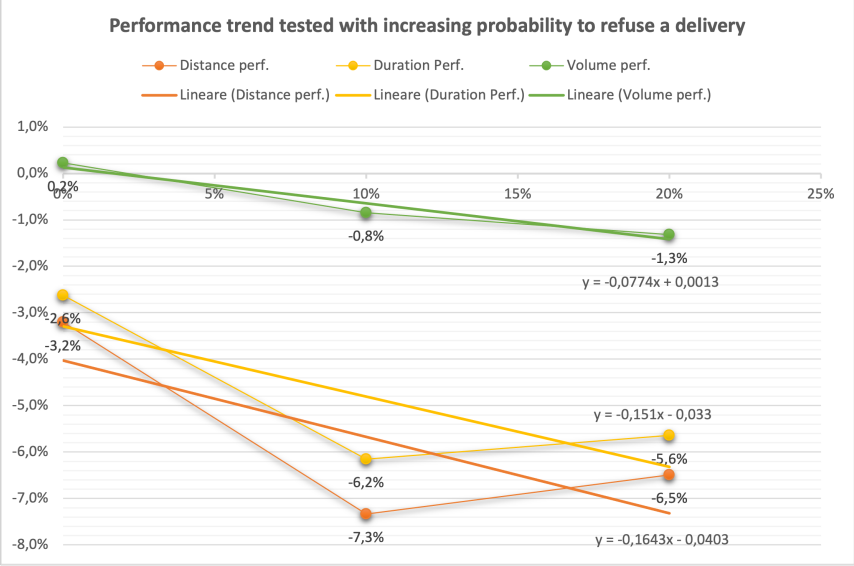

Figure 5.15: Probability to refuse a delivery variation, performance trends.

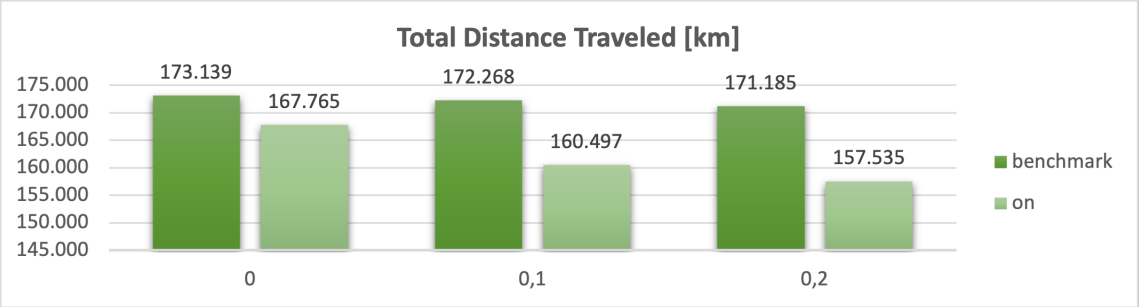

Figure 5.16: Probability to refuse a delivery variation, total travel duration reduction.

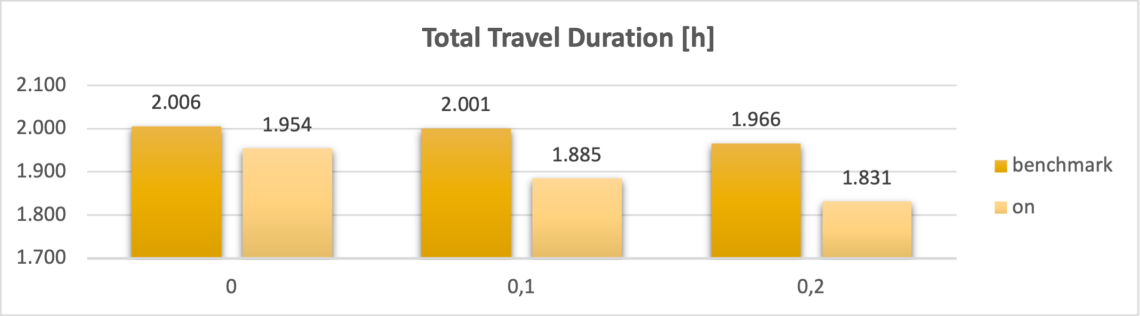

Figure 5.17: Probability to refuse a delivery variation, total distance traveled reduction.

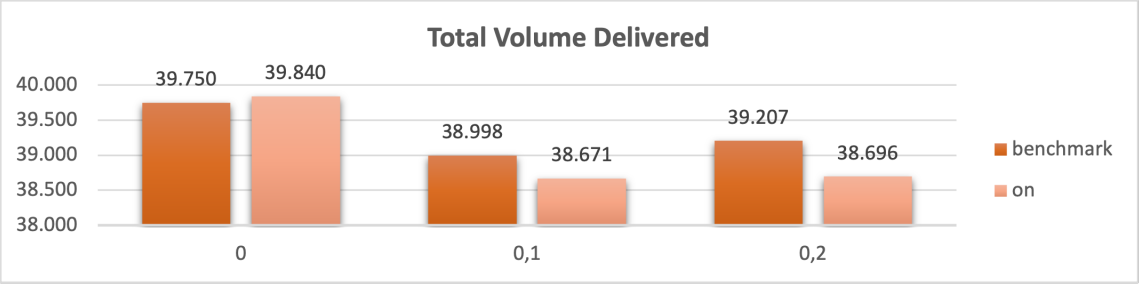

Figure 5.18: Probability to refuse a delivery variation, total volume delivered reduction.

It seems that this type of increasing variability leads to a better exploitation of the likelihood function. This form of stochasticity creates a delay in delivery so

each day there is a bigger customer base to choose from. Despite what we see, it is necessary to specify that a strong assumption has been fixed around this parameter, limiting the maximum number of refusals that a customer can give to one. Further test are needed to establish whether this trend is realistic even in different situations. To sum up, the increasing refuse probability bring to an opposite effect than the cases 3.1 and 3.2, the trends of total distance and duration highlight a better performance. Moreover, it is possible to see a benefit also in the total volume delivered; in this case the trend is evident.

5.3.4 Vehicle capacity variation

Last parameter to be variated is the vehicle capacity. This element is strongly connected to the volume distribution that is fix. We will compare three different scenarios:

- vehicle capacity = 250;
- vehicle capacity = 300;
- vehicle capacity = 350.

The other experimental parameters are set as follow:

- daily orders distribution = Uniform(3,6);
- waiting probability = 10
- refuse probability = 10

Total Distance Traveled [km]

Vehicle Capacity	Benchmark	Optimized	Comparison
250	169.931	164.845	-3,1%
300	172.268	160.497	-7,3%
350	172.378	165.039	-4,4%

Total Volume Delivered

Vehicle Capacity	Benchmark	Optimized	Comparison
250	1.947	1.892	-2,9%
300	2.001	1.885	-6,2%
350	1.995	1.925	-3,7%

Total Volume Delivered

Vehicle Capacity	Benchmark	Optimized	Comparison
250	35.026	34.709	-0,9%
300	38.998	38.671	-0,8%
350	38.321	38.241	-0,2%

Vehicle Capacity	Distance perf.	Duration Perf.	Volume perf.
250	-3,1%	-2,9%	-0,9%
300	-7,3%	-6,2%	-0,8%
350	-4,4%	-3,7%	-0,2%

Figure 5.19: Simulation output with vehicle capacity variation.

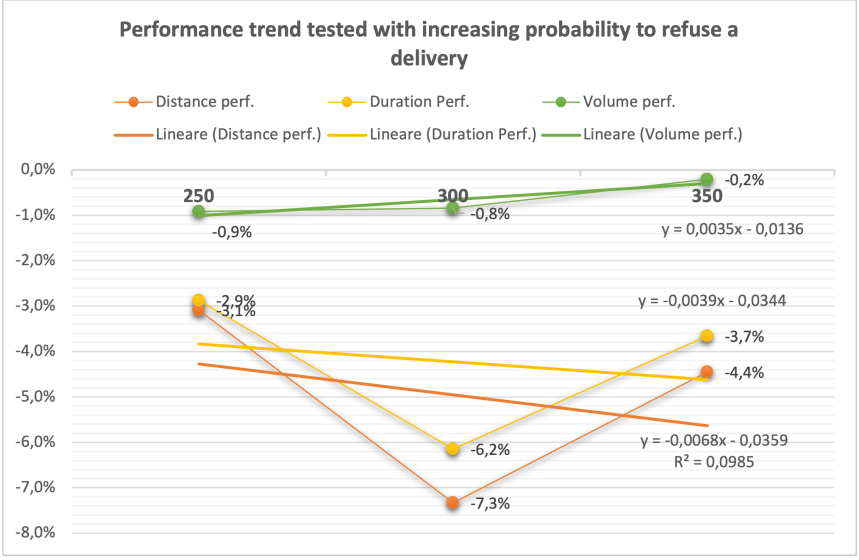

Figure 5.20: Vehicle capacity variation, performance trends.

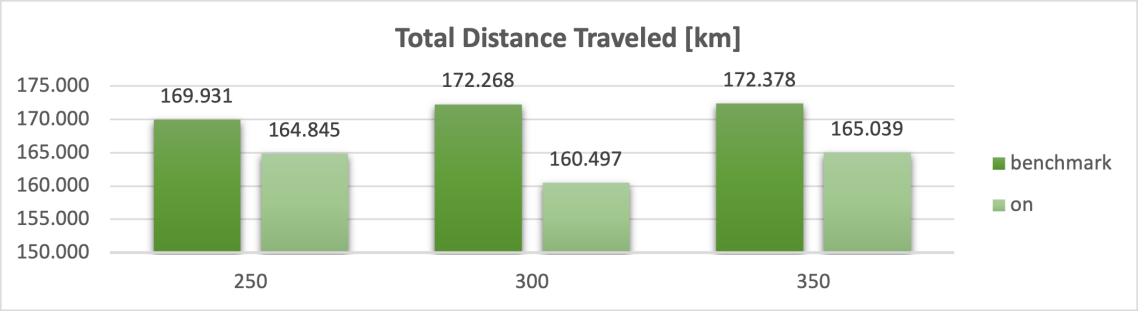

Figure 5.21: Vehicle capacity variation, total travel duration reduction.

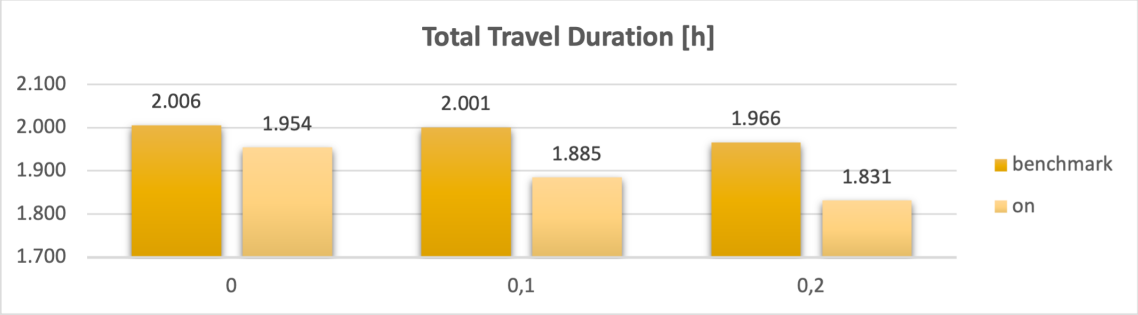

Figure 5.22: Vehicle capacity variation, total distance traveled reduction.

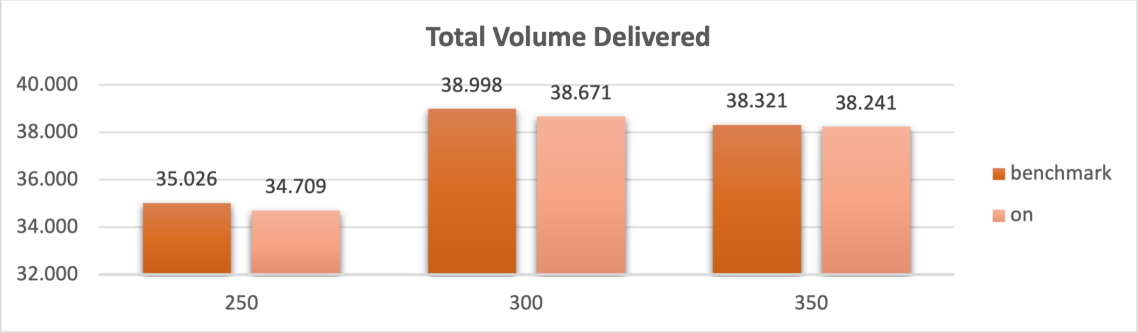

Figure 5.23: Vehicle capacity variation, total volume delivered reduction.

The vehicle capacity reduction brings to a worsening in the performance of the algorithm. We can observe that the more this constraint is strong, the less the

algorithm works well. This is logical because of the nature of this problem. Whenever we have a too large vehicle, we have more freedom to manage the scheduling of delivery because the only strong constraint is the travel time, on the opposite side if the space less the number of daily deliveries will be reduced anyhow. To sum up total distance and duration improve with a larger vehicle. The volume delivered decrease while increasing the volume deliverable, this element prove that the algorithm allows to a better exploitation of the space.

5.4 Observations and weakness

The number of simulations needed to generate this performance analysis is around one hundred, for at least more than 30 hours of run.

It would be interesting to analyse the behaviour of the tool by varying other parameters. At the same time, it would be advisable to evaluate the combined effects of the parameters to highlight any correlations form. Moreover, other simulations can bring to understand what the best setting for the likelihood function is.

Obviously, it would be possible to analyse a lot of other cases, but this tool is actually too much time-spending.

The conclusions that have been drawn must therefore be contextualized under the specific conditions and cannot be considered universal.

We can observe that the tool has good performance, and it is able to solve the study case giving an option for a better management about the delivery scheduling. The likelihood function approach allows to achieve better performance overcoming

the limitation of a human approach (simulated with the dumb solution strategy).

The most interesting scenario is the one where the increased refuse probability brings to better performance. The more complex element to dealt with is the stochasticity in customer behaviour and we can be satisfied to record an improvement according to its variation.

The next chapter will close the discussion opened in the introduction and developed during the chapters about our case scenario. Then will be described possible evolution of this approach and what could be future development around these types of routing problems.

Chapter 6

Conclusion and perspectives

Finally, it is time to take stock of this work.

This paper was born from a theorical topic a complex variant of vehicle routing problem which also include stochasticity and dynamicity.

In a multi-periodical scenario with random customers coming from a large-scale map, we need to decide how to manage deliveries with a limited vehicle capacity. The main aspect is the geographical dimension: we need to decide if delivery to a far location today or wait for nearby orders to be generated.

After an extensive study of the current literature, it was decided to follow the little-travelled path of the simulation.

The simulation was a choice influenced by the real scenario of SMEs. Following a brief investigation, a historical trend emerged in internal management of shipments, not optimal in terms of costs.

Not having the opportunity to start from a real business case, it was decided to

treat the simulation parameters as experimental and conduct test to understand in which simulation the tool behaved better.

From a theorical point of view the main problem of waiting or delivery to far customer was addressed developing a likelihood function able to select cluster of orders to be processed first.

This idea materialized in the writing of a code that was able to manage the scenarios of our interest. The tool was developed with a view to being flexible and allowing even substantial changes in the future to manage situations other than those under analysis. As told before we need to vary the experimental parameters to evaluate the goodness of the likelihood function. The tool is in fact able to simulate and solve different scenario randomly generated according to the set settings.

Many simulations were conducted and by varying some selected parameters it was possible to identify strength and weakness in the solution approach. As expected, an increasing stochasticity implies a performance deterioration, however in some cases the variability brings to improvements. To sum up the likelihood function was able to bring benefits on total travel time and distance travelled in the order of 2% up to 8%. The volume delivered, on the other hand, appeared to have no benefits from this approach.

This thesis work can thus be said to be concluded. Faced with the problem, a simulation software was developed, a solution function was implemented and statistical collections were implemented which allowed to positively evaluate the work carried out.

Despite what has been said, many questions remain open therefore future devel-

opments of this work are manifold.

The likelihood function introduced has been applied to a simulation scenario, the next step should be to apply it to a real dataset. Some constraints have been artificially introduced, future extension may work with multi-periodic delivery scheduling as well with multiple vehicles fleet.

To be even more realistic could be introduced stochastic travel times due to traffic conditions and traffic limitations in cities.

However the more future looking extension would involve replacing the solution algorithm with an AI approach, based on machine learning.

The vehicle routing is actually an open field, it gave possibility to explore new method. Moreover, new technologies like machine learning can lead to a new level of flexibility making of VRP a topic of extreme interest.

Appendix A

Software

First part provides the code that manage the "order" entity. It is treated as a class (OOP) in order to manage multiple object all with the same features. The second is the core of the program, able to manage orders and simulate over multiple days.

A.1 The code, first part: Class Order definition

```
import pandas as pd
import geopandas as gpd
import folium
import random
import datetime
import time
import numpy as np
import openrouteservice as ors
from folium.plugins import BeautifyIcon

# Set graphic pandas dataframe
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

```
# ********************************************************************
#                                                              DATA
# ********************************************************************

# Api key for use open route service
api_key = '*********************************************

ors_client = ors.Client(key=api_key)
# Load data from csv about travel times and coordinates
DATA = 'DATA_Nord-italy-istat,com,lat,lng,reg,prov,sup,numres.cvs'
# COLUMNS: ['istat', 'comune', 'lng', 'lat', 'regione', 'provincia', 'superficie', 'num_residenti
    ']
df_coord = pd.read_csv(DATA, header=0)
df_coord.drop([270], inplace=True)  # Eliminate Torino

# Open the database with the shape of north italy regions
df_region = gpd.read_file('DATA_regioni.geojson')
indexNames0 = df_region[df_region['NOME_REG'] == 'Abruzzo'].index
indexNames1 = df_region[df_region['NOME_REG'] == 'Basilicata'].index
indexNames2 = df_region[df_region['NOME_REG'] == 'Calabria'].index
indexNames3 = df_region[df_region['NOME_REG'] == 'Campania'].index
indexNames4 = df_region[df_region['NOME_REG'] == 'Lazio'].index
indexNames5 = df_region[df_region['NOME_REG'] == 'Marche'].index
indexNames6 = df_region[df_region['NOME_REG'] == 'Molise'].index
indexNames7 = df_region[df_region['NOME_REG'] == 'Puglia'].index
indexNames8 = df_region[df_region['NOME_REG'] == 'Sardegna'].index
indexNames9 = df_region[df_region['NOME_REG'] == 'Sicilia'].index
indexNames10 = df_region[df_region['NOME_REG'] == 'Toscana'].index
indexNames11 = df_region[df_region['NOME_REG'] == 'Umbria'].index
df_region.drop(indexNames0, inplace=True)
df_region.drop(indexNames1, inplace=True)
df_region.drop(indexNames2, inplace=True)
df_region.drop(indexNames3, inplace=True)
df_region.drop(indexNames4, inplace=True)
df_region.drop(indexNames5, inplace=True)
df_region.drop(indexNames6, inplace=True)
df_region.drop(indexNames7, inplace=True)
df_region.drop(indexNames8, inplace=True)
df_region.drop(indexNames9, inplace=True)
df_region.drop(indexNames10, inplace=True)
df_region.drop(indexNames11, inplace=True)

# Define the MAP centered on North Italy
m = folium.Map(location=[45.614037, 10.360107], zoom_start=7)
# Vehicles are located at Torino, plot the depot
depot_name = 'Torino'  # not in use actually
depot = [45.073274, 7.68068748]
```

```
depot_lat = 45.073274
depot_lng = 7.68068748

folium.Marker(
    location=depot,
    tooltip=folium.map.Tooltip("<h4><b>DEPOT </b><b>Istat:\n1272 </b></p><p>Comune di:\nTorino <b
    ></b></p>"),
    icon=folium.Icon(color='green', icon='bus', prefix='fa'),
    # setZIndexOffset=1000
).add_to(m)

# Add region layer to the map
folium.GeoJson(
    df_region,
    name='geojson',
    style_function=lambda feature: {
        'color': 'black',
        'weight': 1,
        'fill': False
    }
).add_to(m)

layer_directions = folium.FeatureGroup(name='layer')  # Add to m directions customer-depot on
    this layer


# ***********************************************************************
#                                                 CLASS DEFINITION " ORDER"
# ***********************************************************************


class Order(object):
    """ A class to model each order in the same way"""
    # SEED SETTING FOR GENERATION
    orders_seed = 111
    random.seed(orders_seed)

    # NOT IN USE Use to randomize the waiting-from-supplier time
    rng2_seed = 13  # SEED WAITING
    rng2 = np.random.RandomState(rng2_seed)

    due_date_delay = 7  # [days] deterministic parameter to set the due date for a new order

    # PARAMETERS
    volume_min = 5
    volume_max = 100
    volume_step = 1
```

```
waiting_min = 1
waiting_max = 5

prob_no_inventory = 0.1  # WAITING PROB. probability for a incoming order to not have
inventory
period = 1  # Day between sequential period

starting_date = datetime.date.today()  # Set the first day of simulation as the "real" today
current_date = starting_date  # Parameter for the simulation, it will change

current_id = 0  # Initialization for the customers id
counter_direction_request = 0  # Count the request to set max 40 per minute

# Creation of a dataframe to store all the info about the orders
df_orders = pd.DataFrame(columns=['customer_ID', 'istat', 'comune', 'distance', 'duration', '
lng', 'lat',
                                  'regione', 'provincia', 'volume', 'data_ordine', 'due_date'
, 'status_ordine',
                                  'release_date', 'data_spedizione', 'route_from_depot'])
df_orders['data_ordine'] = pd.to_datetime(df_orders['data_ordine'], format='%Y%m%d')
df_orders['due_date'] = pd.to_datetime(df_orders['due_date'], format='%Y%m%d')
df_orders['data_spedizione'] = pd.to_datetime(df_orders['data_spedizione'], format='%Y%m%d')
df_orders['release_date'] = pd.to_datetime(df_orders['release_date'], format='%Y%m%d')

# Creation of a dataframe to store all the info about the orders DELIVERED on the FINAL DAY
OF SIMULATION
df_final = pd.DataFrame(columns=['customer_ID', 'istat', 'comune', 'distance', 'duration', '
lng', 'lat',
                                 'regione', 'provincia', 'volume', 'data_ordine', 'due_date',
  'status_ordine',
                                 'release_date', 'data_spedizione'])
df_final['data_ordine'] = pd.to_datetime(df_final['data_ordine'], format='%Y%m%d')
df_final['due_date'] = pd.to_datetime(df_final['due_date'], format='%Y%m%d')
df_final['data_spedizione'] = pd.to_datetime(df_final['data_spedizione'], format='%Y%m%d')
df_final['release_date'] = pd.to_datetime(df_final['release_date'], format='%Y%m%d')

# Creation of a dataframe to store all the info about the orders NOT DELIVERED on the FINAL
DAY OF SIMULATION
df_mid = pd.DataFrame(columns=['customer_ID', 'istat', 'comune', 'distance', 'duration', 'lng
', 'lat',
                               'regione', 'provincia', 'volume', 'data_ordine', 'due_date', '
status_ordine',
                               'release_date', 'data_spedizione'])
df_mid['data_ordine'] = pd.to_datetime(df_mid['data_ordine'], format='%Y%m%d')
df_mid['due_date'] = pd.to_datetime(df_mid['due_date'], format='%Y%m%d')
df_mid['data_spedizione'] = pd.to_datetime(df_mid['data_spedizione'], format='%Y%m%d')
df_mid['release_date'] = pd.to_datetime(df_mid['release_date'], format='%Y%m%d')
```

```
# Creation of a dataframe to store all the travel between depot and any order location
df_travel = pd.DataFrame(
    columns=['Distance', 'Duration', 'From_com.', 'From_prov.', 'To_com.', 'To_prov.', '
To_istat'])

# A list with all the generated routes requested
routes = list()
# A list with all the order created
orders = list()

def __init__(self, customer_id, istat, comune, distance, duration, lng, lat, regione,
provincia,
             volume, data_ordine, due_date, status_ordine, data_spedizione, release_date):
    """ All the information about customers"""
    self.customer_id = customer_id
    self.istat = istat
    self.comune = comune
    self.distance = distance
    self.duration = duration
    self.lng = lng
    self.lat = lat
    self.regione = regione
    self.provincia = provincia
    self.volume = volume
    self.data_ordine = data_ordine
    self.due_date = due_date
    self.status_ordine = status_ordine
    self.release_date = release_date
    self.data_spedizione = data_spedizione
    self.route_from_depot = {}

def show_order_info(self):
    """ Show customer informations"""
    print(f'''
    ID customer: {self.customer_id}
    Municipality istat code: {self.istat}
    Municipality: {self.comune}, {self.provincia}, {self.regione}
    Travel distance (km): {self.distance}
    Travel time (mins): {self.duration}
    Longitude, latidute: {self.lng, self.lat}
    Volume: {self.volume}
    Order date: {self.data_ordine}
    Order due-date: {self.due_date}
    Actual order STATUS: {self.status_ordine}
    Release date (if needed) : {self.release_date}
    Shipment date: {self.data_spedizione}
```

```
    ***''')

def order_generation(self):
    try:
        """ Generation of a single new customer; append all the info to the dataframe"""
        self.customer_id = Order.update_customer_id()
        self.customer_choice_from_database()  # Assign istat, comune, lng, lat, regione,
provincia
        self.compute_route_depot_to_customer()  # Assign distance, duration, route_from_depot
        self.volume = Order.set_volume_ordered()
        self.data_ordine = Order.set_data_ordine()  # Assign the current date
        self.set_due_date()  # Set the due date
        self.status_ordine = Order.set_initial_status_ordine()
        self.set_release_date()  # If no inventory set the release date, and delay the due
date
        self.data_spedizione = ''

        new_order = pd.DataFrame(
            [[self.customer_id, self.istat, self.comune, self.distance, self.duration, self.
lng, self.lat,
              self.regione, self.provincia, self.volume, self.data_ordine, self.due_date,
self.status_ordine,
              self.release_date, self.data_spedizione, self.route_from_depot]],
            columns=['customer_ID', 'istat', 'comune', 'distance', 'duration', 'lng', 'lat',
                     'regione', 'provincia', 'volume', 'data_ordine', 'due_date', '
status_ordine',
                     'release_date', 'data_spedizione', 'route_from_depot'])
        Order.df_orders = pd.concat([Order.df_orders, new_order])
    except:
        print('Distance error from API')


@classmethod
def update_customer_id(cls):
    """ Use to update the customer id for each request"""
    cls.current_id = cls.current_id + 1
    return cls.current_id

def get_customer_id(self):
    """ Use to obtain the number of a order"""
    return self.customer_id

def customer_choice_from_database(self):
    """ Extract a random customer from the database; weight is custom"""
    population = df_coord['istat']
    weights = df_coord['num_residenti']
```

```
        # Extract a random istat using istat population as weight
        random_istat = random.choices(population=population,
                                      weights=weights,
                                      k=1)
        random_customer = random_istat[0]
        extracted_order = df_coord.loc[df_coord['istat'] == random_customer]

        Order.orders.append(extracted_order)
        self.istat = extracted_order.istat.item()
        self.comune = extracted_order.comune.item()
        self.lng = extracted_order.lng.item()
        self.lat = extracted_order.lat.item()
        self.regione = extracted_order.regione.item()
        self.provincia = extracted_order.provincia.item()

    def compute_route_depot_to_customer(self):
        """ Compute the travel distance and duration; then update the travel dataframe"""
        Order.counter_direction_request += 1
        coordinates = ((depot_lng, depot_lat), (self.lng, self.lat))

        direction_params = {'coordinates': coordinates,
                            'profile': 'driving-car',
                            'format_out': 'geojson',
                            'units': 'km',
                            'geometry': 'true'}
        route = ors_client.directions(**direction_params)
        self.route_from_depot = route
        Order.routes.append(route)

        # Avoid to go over 40 requests per minute
        if len(Order.routes) % 39 == 0:
            time.sleep(60)

        distance, duration = route['features'][0]['properties']['summary'].values()
        self.distance = distance
        self.duration = round(duration / 60, 2)

        # Add info about route to the travels information dataframe
        new_record = pd.DataFrame(
            [[self.distance, self.duration, 'Torino', 'TO', self.comune, self.provincia, self.
    istat,
              self.route_from_depot]],
            columns=['Distance', 'Duration', 'From_com.', 'From_prov.', 'To_com.', 'To_prov.',
                     'To_istat', 'route_from_depot'])
        Order.df_travel = pd.concat([Order.df_travel, new_record])

    @classmethod
```

```
    def set_volume_ordered(cls):
        """Use to randomize the volume of a new order"""
        _volume = random.randrange(Order.volume_min, Order.volume_max, step=Order.volume_step)
        return _volume

    @classmethod
    def set_data_ordine(cls):
        """ Use to set the current day of simulation as the order-date"""
        return cls.current_date

    def set_due_date(self):
        """ Use to set the due date of a new order, by a delay sat at the beginning"""
        self.due_date = self.data_ordine + datetime.timedelta(days=Order.due_date_delay)

    @classmethod
    def set_initial_status_ordine(cls):
        """ Use to set the status as already ready to delivery; eventually set the release date
    """
        random_prob = random.random()  # generate a random number between [0.0, 1.0]
        if random_prob <= Order.prob_no_inventory:
            return 'WAITING_FROM_SUPPLIER'
        else:
            return 'READY_TO_DELIVERY'

    def set_release_date(self):
        """ If no inventory, we delay the due date and add a release date"""
        no_inventory_delay = Order.rng2.randint(Order.waiting_min,
                                                Order.waiting_max)  # [days] deterministic
    parameter to set the release date for a new order
        if self.status_ordine == 'WAITING_FROM_SUPPLIER':
            self.due_date = self.due_date + datetime.timedelta(days=no_inventory_delay)
            self.release_date = Order.current_date + datetime.timedelta(days=no_inventory_delay)

    @classmethod
    def change_date(cls):
        """ Use to update by the sat period the date"""
        cls.current_date = cls.current_date + datetime.timedelta(days=Order.period)
        print(f
    """------------------------------------------------------------------------------------------------------------------------

DATE: The date is changed, today is: {cls.current_date}""")

    def update_status_ordine(self):
        """ When happen the a certain date, the order status change"""
        # From WAITING_FROM_SUPPLIER --> READY_TO_DELIVERY
        if self.release_date == Order.current_date:
            self.release_date = ''
```

```
        self.status_ordine = 'READY-TO-DELIVERY'
        print(f"""STATUS: order with ID customer {self.customer_id} is now {self.
status_ordine}.""")

    # From READY-TO-DELIVERY --> BEYOND THE DEADLINE
    if self.release_date == '' and self.status_ordine == 'READY-TO-DELIVERY':
        if self.due_date + datetime.timedelta(days=1) == Order.current_date:
            self.status_ordine = 'BEYOND THE DEADLINE'
            print(f"""STATUS: order with ID customer {self.customer_id} is now {self.
status_ordine}.""")

    # From READY-TO-DELIVERY --> SCHEDULED
    if self.release_date == '' and self.status_ordine == 'READY-TO-DELIVERY':
        if self.data_spedizione != '' \
                and self.data_spedizione == Order.current_date + datetime.timedelta(days=1):
            self.status_ordine = '-SCHEDULED-'
            print(f"""STATUS: order with ID customer {self.customer_id} is now {self.
status_ordine}.""")

    # From SCHEDULED --> DELIVERED
    if self.release_date == '' and self.status_ordine == '-SCHEDULED-':
        if self.data_spedizione == Order.current_date:
            self.status_ordine = 'DELIVERED'
            print(f"""STATUS: order with ID customer {self.customer_id} is now {self.
status_ordine}.""")

    # From BEYOND THE DEADLINE --> DELIVERED BEYOND THE DEADLINE
    elif self.release_date == '' and self.status_ordine == 'BEYOND THE DEADLINE':
        if self.data_spedizione == Order.current_date:
            self.status_ordine = 'DELIVERED BEYOND THE DEADLINE'
            print(f"""STATUS: order with ID customer {self.customer_id} is now {self.
status_ordine}.""")


def set_data_spedizione(self, delay):
    """ Add a shipment date; then update the df_orders. Insert the "delay"."""
    self.data_spedizione = Order.current_date + datetime.timedelta(days=delay)
    print(
        f"""SHIPMENT: Order with ID customer {self.customer_id}. Shipment date scheduled: {
self.data_spedizione}.""")


def map_add_customers_location(self):
    """ Add markers for customer locations to the map"""
    tooltip = folium.map.Tooltip("<h4><b>Customer ID: {}</b>"
                                 "</p><p>Municipality: <b>{}</b></p>"
                                 "</p><p>Order date: <b>{}</b></p>"
```

```
                                    "</p><p>Order due-date: <b>{}</b></p>"
                                    "</p><p>Status: <b>{}</b></p>".format(self.customer_id, self
.cmminomune,
                                                                          self.data_ordine, self
.due_date,
                                                                          self.status_ordine)
                                    )
    folium.Marker(
        location=[self.lat, self.lng],
        tooltip=tooltip,
        icon=BeautifyIcon(
            icon_shape='marker',
            number=int(self.customer_id),
            spin=True,
            text_color='red',
            background_color='#FFF',
            inner_icon_style='font-size:12px;padding-top:-5px;'
        )
    ).add_to(m)

def map_add_customer_direction(self):
    """ Add to the map the directions: from depot to customer"""
    tooltip2 = folium.map.Tooltip("""<h4>Torino --> {0} </h4>
                                    <strong>Duration: </strong>{1:.1f} mins<br>
                                    <strong>Distance: </strong>{2:.3f} km""".format(
        self.comune, self.duration, self.distance))
    folium.GeoJson(
        self.route_from_depot,
        tooltip=tooltip2,
        style_function=lambda feature: {'color': 'blue'}
    ).add_to(layer_directions)

def df_final_creation(self):
    """ Use iterating on a list ORDER_DELIVERED to create a database"""
    last_order_update = pd.DataFrame(
        [[self.customer_id, self.istat, self.comune, self.distance, self.duration, self.lng,
self.lat,
          self.regione, self.provincia, self.volume, self.data_ordine, self.due_date, self.
status_ordine,
          self.release_date, self.data_spedizione]],
        columns=['customer_ID', 'istat', 'comune', 'distance', 'duration', 'lng', 'lat',
                 'regione', 'provincia', 'volume', 'data_ordine', 'due_date', 'status_ordine
',
                 'release_date', 'data_spedizione'])
    Order.df_final = pd.concat([Order.df_final, last_order_update])

def df_mid_creation(self):
```

```
        """ Use iterating on a list order_optimized to create a database"""
        last_order_update = pd.DataFrame(
            [[self.customer_id, self.istat, self.comune, self.distance, self.duration, self.lng,
    self.lat,
              self.regione, self.provincia, self.volume, self.data_ordine, self.due_date, self.
    status_ordine,
              self.release_date, self.data_spedizione]],
            columns=['customer_ID', 'istat', 'comune', 'distance', 'duration', 'lng', 'lat',
                     'regione', 'provincia', 'volume', 'data_ordine', 'due_date', 'status_ordine
    ',
                     'release_date', 'data_spedizione'])
        Order.df_mid = pd.concat([Order.df_mid, last_order_update])
```

A.2 The code, second part: daily routine functions and simulation.

```
import pandas as pd
import random
import numpy as np
import folium
from folium.plugins import BeautifyIcon
import datetime
import openrouteservice as ors
import A06_class as cla

# Set graphic for pandas dataframe
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

api_key = cla.api_key

# ********************************************************************
#                                                          SETTING
# ********************************************************************

""" GENERATION SETTINGS """
TIME_HORIZON = 168
days_of_generation = 1  # Used in n_day_order_creation
PROB_REFUSE_THE_DELIVERY = 0.2  # REFUSE. Is the probability that a customer refuse the delivery
    for a certain date
```

```
daily_min_orders = 3  # Set the MIN number of orders that can be created in a day
daily_max_orders = 6  # Set the MAX number of orders that can be created in a day

# SEED SETTINGS for refuse delivery
rng1_seed = 127
rng1 = np.random.RandomState(rng1_seed)

""" LIKELIHOOD FUNCTION SETTINGS"""
min_distance_likelihood_1_to_2 = 0  # Is a % of distance similarity
min_distance_likelihood_2_to_1 = 0  # Is a % of distance similarity
min_duration_likelihood_1_to_2 = 95  # Is a % of duration similarity (before adjustment)
min_duration_likelihood_2_to_1 = 95  # Is a % of duration similarity (before adjustment)
distance_limit = 0  # Is a limit to choose for optimization only order far enough  NOT IN USE

RTL = 5  # REDUCTION_THRESHOLD_LEVEL Is a % used to adjust the initial level above

MIN_CAPACITY_LOAD = 0.9  # min % level of vehicle load to accept a batch of orders
HALF_CAPACITY_LOAD = 0.75  # half of the vehicle capacity

""" OPTIMIZATION SETTINGS"""
OPTIMIZATION = 'on'  # Set 'on' to use the optimal algorithm instead

on_off_map_creation = 'off'  # write 'on' / 'off' to create daily routing maps; used 2 times in
    delivery_optimization()

NUMBER_OF_VEHICLES = 1  # Max number of vehicle is 3
VEHICLE_CAPACITY = 300
TOT_VEHICLES_CAPACITY = NUMBER_OF_VEHICLES * VEHICLE_CAPACITY

opt_start_hour = 8
opt_end_hour = 23

SERVICE_TIME = 0

""" DATA STRUCTURES """
order_list = list()  # Store ALL the orders
order_ready_to_delivery = list()  # Store orders READY-TO-DELIVERY, so to be scheduled
order_ready_to_optimize = list()  # Store orders TO BE OPTIMIZED for the delivery
order_delivered = list()  # Store orders DELIVERED so not to be touch again

first_refuse = list()  # Store all the refuse, in this way a customer can't refuse multiple time

""" GENERAL COUNTER FOR STATISTICS"""
total_distance = 0  # meters
total_duration = 0  # sec
total_volume = 0
broken_promise = 0  # counter for delivery promised but not happened
```

```
first_sol_counter = 0
second_sol_counter = 0
stupid_avoid_delay = 0
stupid_last_sol = 0
opt_solution_broken = 0  # counter of number of times a good solution is break from a customer
    refuse


# ***********************************************************************
#                                                          FUNCTIONS
# ***********************************************************************


def map_daily_orders_ready():
    """ Use when you want to plot the orders from ready to analyse the algorithm choice"""
    # SETTINGS
    # Load data from csv about travel times and coordinates
    DATA = 'DATA_Nord-italy-istat,com,lat,lng,reg,prov,sup,numres.cvs'
    # COLUMNS: ['istat', 'comune', 'regione', 'lng', 'lat', 'regione', 'provincia', 'superficie',
      'num_residenti']
    df_coord = pd.read_csv(DATA)

    # Vehicles are located at Torino, plot the depot
    depot_name = 'Torino'  # not in use actually
    depot = cla.depot

    # Define the MAP centered on North Italy
    m3 = folium.Map(location=[45.614037, 10.360107], zoom_start=7)

    # Add to m directions customer-depot on this layer
    folium.Marker(
        location=depot,
        tooltip=folium.map.Tooltip("<h4><b>DEPOT </b><b>Istat:\n1272 </b></p><p>Comune di:\
    nTorino <b></b></p>"),
        icon=folium.Icon(color='green', icon='bus', prefix='fa'),
        # setZIndexOffset=1000
    ).add_to(m3)

    layer_directions = folium.FeatureGroup(name='layer')

    for ordine_m in list(order_ready_to_delivery):
        # ADD LOCATIONS
        tooltip = folium.map.Tooltip("<h4><b>Customer ID: {}</b>"
                                     "</p><p>VOLUME: <b>{}</b></p>"
                                     "</p><p>Order date: <b>{}</b></p>"
                                     "</p><p>Order due-date: <b>{}</b></p>"
                                     "</p><p>Status: <b>{}</b></p>".format(ordine_m.customer_id,
```

```
        ordine_m.volume,
                                                                    ordine_m.data_ordine,
        ordine_m.due_date,
                                                                    ordine_m.status_ordine
        )
                                        )
            folium.Marker(
                location=[ordine_m.lat, ordine_m.lng],
                tooltip=tooltip,
                icon=BeautifyIcon(
                    icon_shape='marker',
                    number=int(ordine_m.customer_id),
                    spin=True,
                    text_color='red',
                    background_color='#FFF',
                    inner_icon_style='font-size:12px;padding-top:-5px;'
                )
            ).add_to(m3)

            # ADD DIRECTION
            tooltip2 = folium.map.Tooltip("""<h4>Torino --> {0} </h4>
                                        <strong>Duration: </strong>{1:.1f} mins<br>
                                        <strong>Distance: </strong>{2:.3f} km""".format(
                ordine_m.comune, ordine_m.duration, ordine_m.distance))
            folium.GeoJson(
                ordine_m.route_from_depot,
                tooltip=tooltip2,
                style_function=lambda feature: {'color': 'blue'}
            ).add_to(layer_directions)

        m3.add_child(layer_directions)
        m3.save(f'A07_orders_ready_{cla.Order.current_date + datetime.timedelta(days=1)}.html')


def df_orders_analysis():
    """ Use to analyse the df_orders; so a few statistics about the generated orders"""
    df_input = 'tba'
    df = pd.read_csv(df_input)
    df.drop('Unnamed: 0', axis=1, inplace=True)

    counter = 0

    for row in df.itertuples():
        if row.status_ordine == 'WAITING_FROM_SUPPLIER':
            counter += 1

    volume = df.groupby(by=df.data_ordine).mean()
```

```
    ordini_giornalieri = df.groupby(by=df.data_ordine).count()

    print(f'''
    Sono presenti {len(df)} istanze di cui {counter}, {counter / len(df) * 100}% istanze sono
     WAITING_FROM_SUPPLIER (set 30%)
    Ogni giorno possono esserci tra 2 e 10 ordini; in media sono: {round(ordini_giornalieri['
     istat'].mean(), 2)} (set random in 2, 10)
    Data la media giornaliera dei volumi ordinati, la media delle medie risulta: {round(volume['
     volume'].mean(), 2)} (set unif. 5,100,step=5)
    ''')


def compute_delay():
    days_of_delay = 0
    order_delayed = 0
    for ordine_del in list(order_delivered):
        if ordine_del.status_ordine == 'DELIVERED BEYOND THE DEADLINE' \
                or ordine_del.status_ordine == 'BEYOND THE DEADLINE':
            order_delayed += 1
            specific_delay = ordine_del.data_spedizione - ordine_del.due_date
            days_of_delay += specific_delay.days

    for ordine_ready in list(order_ready_to_delivery):
        if ordine_ready.status_ordine == 'BEYOND THE DEADLINE':
            order_delayed += 1
            specific_delay = cla.Order.current_date - ordine_ready.due_date
            days_of_delay += specific_delay.days

    return days_of_delay, order_delayed


def n_day_orders_creation():
    """ Give him a number of days, for each create a random number of orders between min and max
     """
    for day in range(0, days_of_generation):
        max_daily_orders = random.randint(daily_min_orders, daily_max_orders)
        print(f"\nGENERATION: on the {cla.Order.current_date}, has been generated: {
    max_daily_orders} orders\n")
        for ordine_ in range(0, max_daily_orders):
            ordine_ = cla.Order(int(), int(), str(), float(), float(), float(), float(), str, str
    , int, str, str, str,
                                str, str(''))
            ordine_.order_generation()
            order_list.append(ordine_)


def set_orders_to_ready():
```

```
    """ Look in ORDER_LIST, if an order is ready-to-delivery at it to ORDER_READY_TO_DELIVERY
     list"""
    for ordine_r in list(order_list):
        if ordine_r.status_ordine == 'READY-TO-DELIVERY':
            # if order.data_spedizione == Order.current_date:
            order_ready_to_delivery.append(ordine_r)
            print(f""" Order {ordine_r.customer_id} moved to list ready to delivery""")
            order_list.remove(ordine_r)


def edd_distance_list_sorting(list_name):
    """ Use to sort a list of order based on EARLIEST DUE DATE;
    then all the order with same due_date are sorted from the max distance"""
    list_name.sort(key=lambda x: (x.due_date, -x.duration))


def edd_list_sorting(list_name):
    """ Use to sort a list of order based on EARLIEST DUE DATE"""
    list_name.sort(key=lambda x: x.due_date)


def road_likelihood(order_route_from_depot_0, order_route_from_depot_1):
    """ Used in list_road_likelihood. Give as input two json route, return similarity % about
     distance and duration"""
    # Setting parameters
    distance0, duration0 = order_route_from_depot_0['features'][0]['properties']['summary'].
     values()
    distance1, duration1 = order_route_from_depot_1['features'][0]['properties']['summary'].
     values()
    partial_dist_0 = 0
    partial_dist_1 = 0
    partial_dur_0 = 0
    partial_dur_1 = 0
    stop_signal = 'None'

    # Iteration into dict
    # NB: it stop at the last iteration of the shortest list
    features0 = order_route_from_depot_0['features']
    features1 = order_route_from_depot_1['features']
    for feature0, feature1 in zip(features0, features1):
        segment0 = feature0['properties']['segments']
        segment1 = feature1['properties']['segments']
        for seg0, seg1 in zip(segment0, segment1):
            step0 = seg0['steps']
            step1 = seg1['steps']
            for st0, st1 in zip(step0, step1):
                dist0 = st0['distance']
```

```
            dur0 = st0['duration']
            instr0 = st0['instruction']
            dist1 = st1['distance']
            dur1 = st1['duration']
            instr1 = st1['instruction']

            # Condition to update partial distance and duration
            if stop_signal == 'None':
                if instr0 == instr1 and dist0 == dist1:
                    partial_dist_0 = partial_dist_0 + dist0
                    partial_dist_1 = partial_dist_1 + dist1
                    partial_dur_0 = partial_dur_0 + dur0
                    partial_dur_1 = partial_dur_1 + dur1
                elif instr0 == instr1 and dist0 != dist1:
                    stop_signal = 'FINE SOMIGLIANZA'
                    if dist0 < dist1 and dur0 < dur1:
                        partial_dist_0 = partial_dist_0 + dist0
                        partial_dist_1 = partial_dist_1 + dist0
                        partial_dur_0 = partial_dur_0 + dur0
                        partial_dur_1 = partial_dur_1 + dur0
                    elif dist0 >= dist1 and dur0 >= dur1:
                        partial_dist_0 = partial_dist_0 + dist1
                        partial_dist_1 = partial_dist_1 + dist1
                        partial_dur_0 = partial_dur_0 + dur1
                        partial_dur_1 = partial_dur_1 + dur1
                    elif dist0 >= dist1 and dur0 < dur1:
                        # print("ATTENZIONE: dist0 maggiore di dist1 ma dur0 minore di dis1")
                        # print(dist0, dist1, dur0, dur1)
                        partial_dist_0 = partial_dist_0 + dist1
                        partial_dist_1 = partial_dist_1 + dist1
                        partial_dur_0 = partial_dur_0 + dur0
                        partial_dur_1 = partial_dur_1 + dur0
                    elif dist0 <= dist1 and dur0 > dur1:
                        # print("ATTENZIONE: dist0 minore di dist1 ma dur0 maggiore di dur
1")
                        # print(dist0, dist1, dur0, dur1)
                        partial_dist_0 = partial_dist_0 + dist0
                        partial_dist_1 = partial_dist_1 + dist0
                        partial_dur_0 = partial_dur_0 + dur1
                        partial_dur_1 = partial_dur_1 + dur1

        if partial_dist_0 == partial_dist_1 and partial_dur_0 == partial_dur_1:

            dist_similarity0 = round((partial_dist_0 / distance0) * 100, 2)
            dist_similarity1 = round((partial_dist_1 / distance1) * 100, 2)
            dur_similarity0 = round((partial_dur_0 / duration0) * 100, 2)
            dur_similarity1 = round((partial_dur_1 / duration1) * 100, 2)
```

```
        # print(f"\nDistance similarity: {dist_similarity0}%, {dist_similarity1}%")
        # print(f"\nDuration similarity: {dur_similarity0}%, {dur_similarity1}%")

    else:
        print(f"""ERRORE: il confronto non porta a distanza e durate parziali uguali.
                    part_dur_0 = {partial_dur_0}
                    part_dur_1 = {partial_dur_1}
                    part_dist_0 = {partial_dist_0}
                    part_dist_1 = {partial_dist_1}""")

        dist_similarity0 = round((partial_dist_0 / distance0) * 100, 2)
        dist_similarity1 = round((partial_dist_1 / distance1) * 100, 2)
        dur_similarity0 = round((partial_dur_0 / duration0) * 100, 2)
        dur_similarity1 = round((partial_dur_1 / duration1) * 100, 2)

    return dist_similarity0, dur_similarity0, dist_similarity1, dur_similarity1


def all_list_road_likelihood():
    """ Use on order_ready_to_delivery; obtain a list of comparison about distances and durations
    """
    index = 0
    likelihood_list = list()  # diventa una lista di tuple
    for order_i in order_ready_to_delivery:
        index += 1
        for order_j in order_ready_to_delivery[index:]:
            comparison_results = road_likelihood(order_i.route_from_depot, order_j.
    route_from_depot)
            comparison_info = (order_i.customer_id, order_j.customer_id)
            comparison_results = comparison_results + comparison_info
            if comparison_results not in likelihood_list:
                likelihood_list.append(comparison_results)

    return likelihood_list


def one_to_list_road_likelihood(index):
    """ Use on order_ready_to_delivery; obtain a list of comparison about distances and durations
    """
    likelihood_list = list()  # diventa una lista di tuple
    order_i = order_ready_to_delivery[index]
    for order_j in order_ready_to_delivery[index + 1:]:
        comparison_results = road_likelihood(order_i.route_from_depot, order_j.route_from_depot)
        comparison_info = (order_i.customer_id, order_j.customer_id)
        comparison_results = comparison_results + comparison_info
        if comparison_results not in likelihood_list:
```

```
            likelihood_list.append(comparison_results)

    return likelihood_list


def mean_of_distance_list(list_name):
    """ Return the mean of distance from depot fro a list of orders"""
    tot_distance = 0
    number_of_order = 0

    for ordine_m in list_name:
        number_of_order += 1
        tot_distance = tot_distance + ordine_m.distance

    mean_of_distance = tot_distance / number_of_order
    return mean_of_distance


def single_customer_refuse(customer_order):
    random_prob = rng1.random()  # generate a random number between [0.0, 1.0]
    if customer_order in list(first_refuse):
        return 'not_refused'
    elif random_prob <= PROB_REFUSE_THE_DELIVERY:
        first_refuse.append(customer_order)
        print(f'Order {customer_order.customer_id} REFUSED DELIVERY DATE')
        return 'refused'


def list_customer_refuse(lista, delivery_refused):
    """ Initialize a delivery refused list for the actual day, than update the list whit all the
    refuse"""
    for ordine_c in list(lista):  # pass list order_ready_to_delivery
        if ordine_c not in list(first_refuse):  # check if a customer has already refused a
    delivery
            random_prob = rng1.random()  # generate a random number between [0.0, 1.0]
            if random_prob <= PROB_REFUSE_THE_DELIVERY:
                first_refuse.append(ordine_c)
                delivery_refused.append(ordine_c)
                print(f'Order {ordine_c.customer_id} REFUSED DELIVERY DATE')
                break

    if len(delivery_refused) == 0:
        return 'cluster_accepted'
    else:
        return 'cluster_refused'
```

```
def schedule_choice(delivery_refused, min_capacity_load, threshold_1to2, threshold_2to1):
    """ Is the choice algorithm used in new_set_orders_to_optimize"""
    global opt_solution_broken
    list_index = -1

    for ordine_i in list(order_ready_to_delivery):
        if ordine_i in list(delivery_refused):
            continue

        list_index += 1
        trial_list = list()  # List used inside the choice algorithm

        # Create the likelihood list to choice couples of orders to optimize
        similarity_list = list(one_to_list_road_likelihood(list_index))
        # print(similarity_list)
        for element in list(similarity_list):
            id_2 = element[5]
            dur_1to2 = element[2]
            dur_2to1 = element[3]
            # First choice condition
            if dur_1to2 > threshold_1to2 and dur_2to1 > threshold_2to1:
                # Search the second order and add both to a trial list
                for ordine_j in list(order_ready_to_delivery):
                    if ordine_j.customer_id == id_2:
                        if ordine_i not in trial_list:
                            trial_list.append(ordine_i)
                        if ordine_j not in trial_list:
                            trial_list.append(ordine_j)
                        break

        # Update capacity counter for verify condition
        capacity_counter = 0
        for ordine_t in list(trial_list):
            capacity_counter += ordine_t.volume

        # Check capacity condition, then check possible deliveries refused
        if TOT_VEHICLES_CAPACITY * min_capacity_load < capacity_counter < TOT_VEHICLES_CAPACITY:
            # call customers from trial list, if one refuse repeat all
            answer = list_customer_refuse(trial_list, delivery_refused)

            if answer == 'cluster_refused':
                opt_solution_broken += 1

            if answer == 'cluster_accepted':
                order_ready_to_optimize.extend(trial_list)
                # Remove orders from ready to delivery list
                for j in list(trial_list):
```

```
                    print(f""" Order {j.customer_id} moved to list ready to optimize""")
                    if j in list(order_ready_to_delivery):
                        order_ready_to_delivery.remove(j)

                return 'ok'
            else:
                continue
        return 'no'


def opt_set_orders_to_optimize():
    """ Is the function used to choose orders to schedule for tomorrow"""
    global broken_promise, first_sol_counter, second_sol_counter, stupid_avoid_delay,
     stupid_last_sol
    # Initialize the list ready adding orders that stay in optimize from yesterday
    order_ready_to_delivery.extend(order_ready_to_optimize)
    for x in list(order_ready_to_optimize):
        print(f'ATTENZIONE: ordine {x.customer_id} promesso ma non schedulato ieri')
        broken_promise += 1
        order_ready_to_optimize.remove(x)
    edd_distance_list_sorting(order_ready_to_delivery)  # Sort with EDD + DISTANCE orders ready
     to be scheduled

    stop_signal = ''
    last_call_orders = list()
    delivery_refused = list()

    # Use opt stupid solution function to avoid delay
    for ordine_l in list(order_ready_to_delivery):
        if (ordine_l.due_date == cla.Order.current_date + datetime.timedelta(days=1)
            or ordine_l.due_date == cla.Order.current_date + datetime.timedelta(days=2)
            or ordine_l.status_ordine == 'BEYOND THE DEADLINE') \
                and ordine_l.data_spedizione == '':
            last_call_orders.append(ordine_l)

    if len(last_call_orders) > 0:
        print(f"OPT: use dumb solution to avoid delay")
        stupid_avoid_delay += 1
        avoid_delay_stupid_solution()

    else:
        # todo RITENGO POSSIBILE MIGLIORARE ULTERIORMENTE TALI CONDIZIONI
        # Look for a first solution
        threshold_1to2 = min_duration_likelihood_1_to_2
        threshold_2to1 = min_duration_likelihood_2_to_1
        if len(order_ready_to_optimize) == 0:
            for i in range(0, 19):
```

```
            if stop_signal == 'stop':
                print(f"OPT: first best solution found with {threshold_1to2}% and {
threshold_2to1}%")
                first_sol_counter += 1
                break
            else:
                threshold_1to2 = min_duration_likelihood_1_to_2 - RTL * i
                threshold_2to1 = min_duration_likelihood_2_to_1 - RTL * i
                capacity_threshold = MIN_CAPACITY_LOAD
                while threshold_1to2 >= 0 and stop_signal != 'stop':
                    answer = schedule_choice(delivery_refused, capacity_threshold,
threshold_1to2, threshold_2to1)
                    threshold_1to2 -= RTL
                    if answer == 'ok':
                        stop_signal = 'stop'

    print('segnale 1')

    # If a first solution in not found, reduce vehicle capacity limit and search another
    if len(order_ready_to_optimize) == 0:
        print(f"OPT: first choice solution not available. Check if 0 = {len(
order_ready_to_optimize)}")
        threshold_1to2 = min_duration_likelihood_1_to_2
        threshold_2to1 = min_duration_likelihood_2_to_1
        for i in range(0, 19):
            if stop_signal == 'stop':
                print(
                    f"OPT: second best solution found. Orders number: {len(
order_ready_to_optimize)} with {threshold_1to2}% and {threshold_2to1}%")
                second_sol_counter += 1
                break
            else:
                threshold_1to2 = min_duration_likelihood_1_to_2 - RTL * i
                threshold_2to1 = min_duration_likelihood_2_to_1 - RTL * i
                capacity_threshold = HALF_CAPACITY_LOAD
                while threshold_1to2 >= 0 and stop_signal != 'stop':
                    answer = schedule_choice(delivery_refused, capacity_threshold,
threshold_1to2, threshold_2to1)
                    threshold_1to2 -= RTL
                    if answer == 'ok':
                        stop_signal = 'stop'

    print('segnale 2')

    if len(order_ready_to_optimize) == 0:
        print(f"OPT: second best solution not available. Check if 0 = {len(
order_ready_to_optimize)}")
```

```
            print('OPT: FAILED, using dumb solution')
            stupid_last_sol += 1
            opt_dumb_solution(delivery_refused)

    print(f'Number of delivery refused today: {len(delivery_refused)}')


def avoid_delay_stupid_solution():
    """ Is the solution to avoid delay """
    # Initialize the list ready adding orders that stay in optimize from yesterday
    edd_list_sorting(order_ready_to_delivery)  # Sort with EDD orders ready to be scheduled

    capacity_counter = 0
    delivery_accepted = list()
    delivery_refused = list()

    # Choice the orders simply following the order edd + max duration
    if len(order_ready_to_delivery) >= 2:
        for ordine_e in list(order_ready_to_delivery):
            answer = single_customer_refuse(ordine_e)
            if answer == 'refused':
                delivery_refused.append(ordine_e)
            else:
                if capacity_counter + ordine_e.volume > TOT_VEHICLES_CAPACITY:
                    print(
                        f'STOP, limite capacit  raggiunto: {capacity_counter + ordine_e.volume}
    > {TOT_VEHICLES_CAPACITY}')
                    break
                else:
                    capacity_counter += ordine_e.volume
                    delivery_accepted.append(ordine_e)

    # Remove orders chosen for optimization from order_ready_to_delivery
    for ordine_a in list(delivery_accepted):
        if ordine_a in order_ready_to_delivery:
            order_ready_to_delivery.remove(ordine_a)
        if ordine_a not in order_ready_to_optimize:
            print(f""" Order {ordine_a.customer_id} moved to list ready to optimize""")
            order_ready_to_optimize.append(ordine_a)


def opt_dumb_solution(delivery_refused_list):
    """ Last call solution when has not been possible find an optimal one """
    # Initialize the list ready adding orders that stay in optimize from yesterday
    edd_list_sorting(order_ready_to_delivery)  # Sort with EDD orders ready to be scheduled

    capacity_counter = 0
```

```
    delivery_accepted = list()

    # Choice the orders simply following the order edd + max duration
    if len(order_ready_to_delivery) >= 2:
        for ordine_e in list(order_ready_to_delivery):
            if ordine_e in delivery_refused_list:
                continue
            else:
                answer = single_customer_refuse(ordine_e)
                if answer == 'refused':
                    delivery_refused_list.append(ordine_e)
                else:
                    if capacity_counter + ordine_e.volume > TOT_VEHICLES_CAPACITY:
                        print(
                            f'STOP, limite capacit  raggiunto: {capacity_counter + ordine_e.
    volume} > {TOT_VEHICLES_CAPACITY}')
                        break
                    else:
                        capacity_counter += ordine_e.volume
                        delivery_accepted.append(ordine_e)

    # Remove orders chosen for optimization from order_ready_to_delivery
    for ordine_a in list(delivery_accepted):
        if ordine_a in order_ready_to_delivery:
            order_ready_to_delivery.remove(ordine_a)
        if ordine_a not in order_ready_to_optimize:
            print(f""" Order {ordine_a.customer_id} moved to list ready to optimize""")
            order_ready_to_optimize.append(ordine_a)


def dumb_solution():
    """ Simple solution for benchmark; run when OPTIMIZATION is not 'on' """
    global broken_promise
    # Initialize the list ready adding orders that stay in optimize from yesterday
    order_ready_to_delivery.extend(order_ready_to_optimize)
    for x in list(order_ready_to_optimize):
        print(f'ATTENZIONE: ordine {x.customer_id} promesso ma non schedulato ieri')
        broken_promise += 1
        order_ready_to_optimize.remove(x)
    edd_list_sorting(order_ready_to_delivery)  # Sort with EDD orders ready to be scheduled

    capacity_counter = 0
    delivery_accepted = list()
    delivery_refused = list()

    # Choice the orders simply following the order edd + max duration
    for ordine_e in list(order_ready_to_delivery):
```

```
            answer = single_customer_refuse(ordine_e)
            if answer == 'refused':
                delivery_refused.append(ordine_e)
            else:
                if capacity_counter + ordine_e.volume > TOT_VEHICLES_CAPACITY:
                    print(
                        f'STOP, limite capacit  raggiunto: {capacity_counter + ordine_e.volume} > {
    TOT_VEHICLES_CAPACITY}')
                    break
                else:
                    capacity_counter += ordine_e.volume
                    delivery_accepted.append(ordine_e)

        # Remove orders chosen for optimization from order_ready_to_delivery
        for ordine_a in list(delivery_accepted):
            if ordine_a in order_ready_to_delivery:
                order_ready_to_delivery.remove(ordine_a)
            if ordine_a not in order_ready_to_optimize:
                print(f""" Order {ordine_a.customer_id} moved to list ready to optimize""")
                order_ready_to_optimize.append(ordine_a)


    def delivery_optimization():
        """ Read a list of orders and optimize. Create a map for each day where an optimization is
        done"""
        global total_distance, total_duration, total_volume
        start_hour = opt_start_hour
        end_hour = opt_end_hour
        deliveries_data = order_ready_to_optimize

        current_date = cla.Order.current_date + datetime.timedelta(days=1)
        time_start_work = datetime.time(start_hour, 0, 0)
        time_end_work = datetime.time(end_hour, 0, 0)
        start_hour = datetime.datetime.combine(current_date, time_start_work)  # Fri 8-20:00,
        expressed in POSIX timestamp
        end_hour = datetime.datetime.combine(current_date, time_end_work)

        if len(deliveries_data) > 1:

            # SETTINGS
            # Define the map centered on North Italy
            # Vehicles are located at Torino, plot the depot
            depot = [45.073274, 7.68068748]
            depot_lat = 45.073274
            depot_lng = 7.68068748
            m2 = folium.Map(location=[45.614037, 10.360107], zoom_start=7.2)
```

```
    # Plot the locations on the map with more info in the ToolTip
    if on_off_map_creation == 'on':
        for location in list(deliveries_data):
            tooltip = folium.map.Tooltip("<h4><b>ID {}</b></p><p>Volume: <b>{}</b></p>".
format(
                location.customer_id, location.volume
            ))
            folium.Marker(
                location=[location.lat, location.lng],
                tooltip=tooltip,
                icon=BeautifyIcon(
                    icon_shape='marker',
                    number=int(location.customer_id),
                    spin=True,
                    text_color='red',
                    background_color="#FFF",
                    inner_icon_style="font-size:12px;padding-top:-5px;"
                )
            ).add_to(m2)

        # Plot the depot
        folium.Marker(
            location=depot,
            tooltip=folium.map.Tooltip(
                "<h4><b>DEPOT </b><b>Istat:\n1272 </b></p><p>Comune di:\nTorino <b></b></p>")
,
            icon=folium.Icon(color='green', icon='bus', prefix='fa'),
            setZIndexOffset=1000
        ).add_to(m2)
        m2.save(f'A07_optimization_mappa_{cla.Order.current_date + datetime.timedelta(days=1)
}.html')

    # VEHICLE ROUTING SET UP

    # Define the vehicles
    vehicles = list()
    for idx in range(NUMBER_OF_VEHICLES):
        vehicles.append(
            ors.optimization.Vehicle(
                id=idx,
                start=list([depot_lng, depot_lat]),
                end=list([depot_lng, depot_lat]),
                capacity=[VEHICLE_CAPACITY],
                time_window=[int(start_hour.strftime("%s")), int(end_hour.strftime("%s"))]
            )
        )
```

```
    # Define the delivery station
    deliveries = list()
    for delivery in list(deliveries_data):
        deliveries.append(
            ors.optimization.Job(
                id=delivery.customer_id,
                location=[delivery.lng, delivery.lat],  # Expected order is Longitude,
Latitude
                service=SERVICE_TIME,  # Assume 20 minutes at each site as service time
                amount=[delivery.volume]
            )
        )

    # Initialize a client and make the request
    ors_client = ors.Client(key=api_key)
    result = ors_client.optimization(
        jobs=deliveries,
        vehicles=vehicles,
        geometry=True
    )

    # Add the output to the map
    if on_off_map_creation == 'on':
        for color, route in zip(['green', 'red', 'blue'], result['routes']):
            decoded = ors.convert.decode_polyline(route['geometry'])
            gj = folium.GeoJson(
                name='Vehicle {}'.format(route['vehicle']),
                data={"type": "FeatureCollection", "features": [{"type": "Feature",
                                                                 "geometry": decoded,
                                                                 "properties": {"color":
color}
                                                                 }]},
                style_function=lambda x: {"color": x["properties"]['color']}
            )
            gj.add_child(folium.Tooltip(
                """<h4>Vehicle {vehicle}</h4>
                <b>Distance</b> {distance} m <br>
                <b>Duration</b> {duration} secs
                """.format(**route)
            ))
            gj.add_to(m2)

        folium.LayerControl().add_to(m2)
        m2.save(f'A07_optimization_mappa_{cla.Order.current_date + datetime.timedelta(days=1)
}.html')

    # DATA VIEW
```

```
# Extract relevant field from the response
extract_field = ['distance', 'duration', 'amount']
data = [{key: route[key] for key in extract_field} for route in result['routes']]

# Update general statistics
for d in list(data):
    total_distance = total_distance + d['distance']
    total_duration = total_duration + d['duration']
    total_volume = total_volume + d['amount'][0]

vehicles_df = pd.DataFrame(data)
vehicles_df.index.name = 'vehicle'

# Create a list to display the schedule for all the vehicles
stations = list()
delivered_id = list()
for route in result['routes']:
    vehicle = list()
    for step in route['steps']:
        delivered_id.append(step.get('id'))
        vehicle.append(
            [
                step.get("job", "Depot"),  # Station ID
                step["arrival"],  # Arrival time
                step["arrival"] + step.get("service", 0)  # Departure time
            ]
        )
    stations.append(vehicle)

df_all_tour = pd.DataFrame()

for route in range(len(vehicles_df)):
    route = pd.DataFrame(stations[route], columns=["Station ID", "Arrival", "Departure"])
    route['Arrival'] = pd.to_datetime(route['Arrival'], unit='s')
    route['Departure'] = pd.to_datetime(route['Departure'], unit='s')
    df_all_tour = pd.concat([df_all_tour, route])

""" From a list of numbers, obtained from optimization, add orders to order_delivered"""
# Add delivered order to order_delivered list
for idx in delivered_id:
    for ordine_h in list(order_ready_to_optimize):
        if ordine_h.customer_id == idx:
            ordine_h.set_data_spedizione(1)
            order_delivered.append(ordine_h)
            order_ready_to_optimize.remove(ordine_h)

print(vehicles_df)
```

```
        print(df_all_tour)


def all_list_update():
    """ UPDATE orders status in all lists """
    for ordine1 in order_list:
        ordine1.update_status_ordine()
    for ordine2 in order_ready_to_delivery:
        ordine2.update_status_ordine()
    for ordine3 in order_ready_to_optimize:
        ordine3.update_status_ordine()
    for ordine4 in order_delivered:
        ordine4.update_status_ordine()


def daily_routine():
    """ A single day routine:
            1. change date
            2. create new orders
            3. update orders and set the order_list
            4. choose orders to be scheduled next day
            5. optimize orders
            """

    cla.Order.change_date()  # Change date
    n_day_orders_creation()  # Creation of orders; automatically stored in order_list
    all_list_update()  # UPDATE orders status in all lists
    set_orders_to_ready()  # Put ready-to-delivery orders in order_ready_to_delivery

    if on_off_map_creation == 'on':
        map_daily_orders_ready()

    if OPTIMIZATION == 'on':
        opt_set_orders_to_optimize()
    else:
        dumb_solution()

    delivery_optimization()  # Take from order_ready_to_optimize and made the optimization
    all_list_update()  # UPDATE orders status in all lists


# ********************************************************************
#                                                        MAIN()
# ********************************************************************

def main():
    """ X days simulations """
```

```
for a in range(0, TIME_HORIZON):
    daily_routine()

# DF_ORDERS is the collection of all the orders created on the day of their creation
for ordine in order_delivered:
    ordine.df_final_creation()

# DF_MID is the collection of all the order from ready_to_optimize and ready_to_delivery
order_ready_to_delivery.extend(order_ready_to_optimize)
for ordine in order_ready_to_delivery:
    ordine.df_mid_creation()

total_order_delay = compute_delay()[1]
total_days_delay = compute_delay()[0]

cla.Order.df_orders.to_csv('A07_df_orders.csv')
cla.Order.df_mid.to_csv('A07_df_mid.csv')
cla.Order.df_final.to_csv('A07_df_final.csv')

print('\nALL ORDERS ************************************************************')
print(cla.Order.df_orders)
print('\nREADY-TO-DELIVERY (LAST DAY) ******************************************')
print(cla.Order.df_mid)
print('\nDELIVERED (LAST DAY) **************************************************')
print(cla.Order.df_final)
print('************************************************************************')

print(f"""
OPTIMIZATION: {OPTIMIZATION}

ORDINI DELIVERED: {len(cla.Order.df_final)}
ORDINI GENERATI: {len(cla.Order.df_orders)}
    di cui ready (last day): {len(order_ready_to_delivery)}
    di cui in optimize (last day): {len(order_ready_to_optimize)}
    di cui in waiting-from-supplier (last day): {len(order_list)}

NUMERO ORDINI IN RITARDO (on last day): {total_order_delay}
GIORNI TOTALI DI RITARDO (on last day): {total_days_delay}

Opt solution broken from refuses: {opt_solution_broken}

RIFIUTI TOTALI: {len(first_refuse)}
ORDINI PROMESSI NON CONSEGNATI: {broken_promise}

TOTALE DISTANZA COPERTA: {round(total_distance / 1000, 2)} kms
TOTALE DURATA VIAGGIO: {round(total_duration / 3600, 2)} hours
TOTALE VOLUME CONSEGNATO: {total_volume}
```

```
    SIMULATION PERIOD: {TIME_HORIZON} days
        from {cla.Order.starting_date} to {cla.Order.current_date}

    Counter solution found: {first_sol_counter + second_sol_counter + stupid_last_sol +
    stupid_avoid_delay}
        - first_solution: {first_sol_counter}
        - second_solution: {second_sol_counter}
        - last_call: {stupid_last_sol}
        - avoid_delay: {stupid_avoid_delay}

    Order generation from UNIFORM({daily_min_orders}, {daily_max_orders})
    Volume of orders from UNIFORM({cla.Order.volume_min}, {cla.Order.volume_max}, step={cla.Order
    .volume_step})
    Waiting from supplier delay from UNIFORM({cla.Order.waiting_min}, {cla.Order.waiting_max})

    Waiting from supplier prob: {cla.Order.prob_no_inventory * 100}%
    Due date delay: {cla.Order.due_date_delay} days
    Refuse prob: {PROB_REFUSE_THE_DELIVERY * 100}%

    Veicoli: {NUMBER_OF_VEHICLES}
    Capacit  veicolo: {VEHICLE_CAPACITY}
    Service time: {SERVICE_TIME}
    Working daily time: {opt_start_hour - 1}:00 - {opt_end_hour - 1}:00
    Start_road_likelihood: {min_duration_likelihood_1_to_2}% and {min_duration_likelihood_2_to_1
    }%
    REDUCTION_THRESHOLD_LEVEL: -{RTL}%
    First_solution capacity threshold: {MIN_CAPACITY_LOAD * 100}%
    Second_solution capacity threshold: {HALF_CAPACITY_LOAD * 100}%

    Seed orders: {cla.Order.orders_seed}
    Seed refuse (rng1): {rng1_seed}
    Seed delay (rng2): {cla.Order.rng2_seed}


    CHECK IF : {len(order_list) + len(cla.Order.df_final) + len(cla.Order.df_mid)} = {len(cla.
    Order.df_orders)}
    """)


if __name__ == '__main__':
    main()
```

Bibliography

[1] Campbell, Ann Melissa, and Wilson, Jill Hardin. *Forty Years of Periodic Vehicle Routing.* Networks, vol. 63, no. 1, 2014, pp. 2–15.

[2] Clarke G. and J.W. Wright *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Research, vol. 12, no. 4, 1964, pp.568-581.

[3] Dantzig, G. B, and Ramser, J. H. *The Truck Dispatching Problem.* Management Science, vol. 6, no. 1, 1959, pp. 80–91.

[4] Mor, A, and Speranza, M. G. *Vehicle Routing Problems over Time: a Survey.* 4OR, vol. 18, no. 2, 2020, pp. 129–149.

[5] Psaraftis, Harilaos N, et al. *Dynamic Vehicle Routing Problems: Three Decades and Counting.* Networks, vol. 67, no. 1, 2016, pp. 3–31.

[6] Dempster, M. A. H, et al. *Analytical Evaluation of Hierarchical Planning Systems.* Operations Research, vol. 29, no. 4, 1981, pp. 707–716.

[7] Ritzinger, Ulrike, et al. *A Survey on Dynamic and Stochastic Vehicle Routing Problems.* International Journal of Production Research, vol. 54, no. 1, 2016, pp. 215–231.

[8] Archetti, Claudia, et al. *Multi-Period Vehicle Routing Problem with Due Dates.* Computers Operations Research, vol. 61, 2015, pp. 122–134.

[9] Archetti, Claudia, et al. *A Branch-and-Cut Algorithm for a Vendor-Managed Inventory-Routing Problem.* Transportation Science, vol. 41, no. 3, 2007, pp. 382–391.

[10] Oyola, Jorge, et al. *The Stochastic Vehicle Routing Problem, a Literature Review, Part I: Models.* EURO Journal on Transportation and Logistics, vol. 7, no. 3, 2018, pp. 193–221.

[11] Adewumi, Aderemi Oluyinka, et al. *A Survey of Recent Advances in Vehicle Routing Problems.* International Journal of System Assurance Engineering and Management, vol. 9, no. 1, 2018, pp. 155–172.

[12] Sarasola, Briseida, et al. *Variable Neighborhood Search for the Stochastic and Dynamic Vehicle Routing Problem.* Annals of Operations Research, vol. 236, no. 2, 2016, pp. 425–461.

[13] Braekers, Ramaekers, Van Nieuwenhuyse, Inneke. *The vehicle routing problem: state of art classification and review.* Computers Industrial Engineering, vol. 99, 2016, pp. 300-313.

[14] Bruck, Bruno P, et al. *A Decision Support System for Attended Home Services.* Interfaces (Providence), vol. 50, no. 2, 2020, pp. 137–152.

[15] Vidal, Thibaut, et al. *A Concise Guide to Existing and Emerging Vehicle Routing Problem Variants.* European Journal of Operational Research, vol. 286, no. 2, 2020, pp. 401–416.

[16] Coelho, Leandro C, et al. *Heuristics for Dynamic and Stochastic Inventory-Routing.* Computers Operations Research, vol. 52, 2014, pp. 55–67.

[17] Han, Shuihua, et al. *Appointment Scheduling and Routing Optimization of Attended Home Delivery System with Random Customer Behavior.* European Journal of Operational Research, vol. 262, no. 3, 2017, pp. 966–980.

[18] Bent, Russell W, and Van Hentenryck, Pascal. *Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers.* Operations Research, vol. 52, no. 6, 2004, pp. 977–987.

[19] Restrepo, Maria I, et al. *Integrated Shift Scheduling and Load Assignment Optimization for Attended Home Delivery.* Transportation Science, vol. 53, no. 4, 2019, pp. 1150–1174.

[20] Archetti, Claudia, et al. *A Branch-and-Cut Algorithm for a Vendor-Managed Inventory-Routing Problem.* Transportation Science, vol. 41, no. 3, 2007, pp. 382–391.

[21] Campbell, Ann Melissa, and Wilson, Jill Hardin. *Forty Years of Periodic Vehicle Routing.* Networks, vol. 63, no. 3, 2014, p. 276.

[22] https://www.istat.it/it/archivio/238337

[23] https://ec.europa.eu/growth/smes/sme-definition$_en$

[24] https://www.pmi.it/tecnologia/prodotti-e-servizi-ict/330571/commercio-la-nuova-gestione-degli-ordini.html

[25] https://ec.europa.eu/eurostat/web/digital-economy-and-society/data/database

[26] https://www.lastampa.it/economia/2020/09/09/news/l-e-commerce-ha-fatto-boom-71-grazie-al-covid-1.39282548

[27] https://www.leadershipmanagementmagazine.com/articoli/vincoli-alloutsourcing-logistico-le-pmi-italiane/