Development of a delivery process simulation model with a focus on Italy

TESI DI LAUREA MAGISTRALE IN MANAGEMENT ENGINEERING
INGEGNERIA GESTIONALE

Author: **Riccardo Colombelli**

Student ID: 994911
Advisor: Riccardo Mangiaracina
Co-advisor: Arianna Seghezzi, Chiara Siragusa
Academic Year: 2022-23

Abstract

E-commerce is increasingly becoming a part of everyone's life, evolving into a nearly habitual practice, particularly in Italy. This continuous growth has significantly impacted the performance of the entire delivery process, with a pronounced effect on the last-mile. Therefore, it is crucial to be able to study the entire delivery process to analyze and enhance its efficiency in the most critical parts.

The aim of this thesis was to develop a model, with a focus on the Italian context, capable of simulating and modeling all three phases that constitute the delivery process: first-mile, last-mile, and pick-up.

To achieve a simulation as faithful as possible to reality, a thorough investigation was conducted on the facilities of the major couriers operating in the Italian market looking for information regarding the type, quantity, and, most importantly, the location of warehouses and hubs. This effort aimed to create an updated and precise database to serve as the foundation for the program's operation.

The program was crafted with a focus on simplicity and adaptability, ensuring its usability across diverse stakeholders. Its flexibility stands out as a key asset, making it applicable to a wide range of users, including students and company owners. Importantly, it serves as a versatile foundation, designed to accommodate various updates and modifications over time. This adaptability allows the program to evolve and meet different objectives as it is utilized.

Key-words: e-commerce, delivery process, simulation modeling, Italian courier services

Abstract in italiano

L'e-commerce sta diventando sempre più parte della vita di tutti, evolvendosi in una pratica quasi abituale, soprattutto in Italia. Questa crescita continua ha impattato significativamente le prestazioni dell'intero processo di consegna, con un effetto pronunciato sul last-mile. Pertanto, è cruciale essere in grado di studiare l'intero processo di consegna per analizzarne ed aumentarne l'efficienza nelle parti più critiche.

Lo scopo di questa tesi era sviluppare un modello, con un focus sul contesto italiano, in grado di simulare e modellare tutte e tre le fasi che costituiscono il processo di consegna: first-mile, last-mile e pick-up.

Per ottenere una simulazione il più fedele possibile alla realtà, è stata condotta un'indagine approfondita sulle strutture dei principali corrieri operanti nel mercato italiano, cercando informazioni riguardo al tipo, alla quantità e, soprattutto, alla posizione di magazzini e hub. Questo sforzo aveva l'obiettivo di creare un database aggiornato e preciso per servire da base per il funzionamento del programma.

Il programma è stato progettato con un'attenzione alla semplicità e all'adattabilità, garantendo la sua usabilità tra diversi stakeholder. La sua flessibilità si distingue come un punto di forza, rendendolo applicabile a una vasta gamma di utenti, tra cui studenti e proprietari di aziende. In modo significativo, funge da fondamento versatile, progettato per accogliere vari aggiornamenti e modifiche nel tempo. Questa adattabilità consente al programma di evolversi e raggiungere diversi obiettivi durante il suo utilizzo.

Parole chiave: e-commerce, processo di consegna, modello di simulazione, servizi di corrieri italiani

Contents

1 Introduction

1.1. E-commerce

1.1.1. E-commerce growth

E-commerce has emerged as a pivotal player in the global market, steadily gaining prominence across both mature and emerging economies (Mangiaracina *et al.*, 2019). 2021 witnessed a significant milestone in this realm, with retail e-commerce sales surging to a staggering 5.2 trillion U.S. dollars worldwide. This figure, noteworthy, is projected to experience a remarkable 56% increase in the coming years, soaring to an estimated 8.1 trillion dollars by 2026 (eMarketer, 2022). This meteoric rise underscores the crucial role that e-commerce plays in shaping contemporary economies, fundamentally altering the landscape of consumer transactions and supply chains alike.

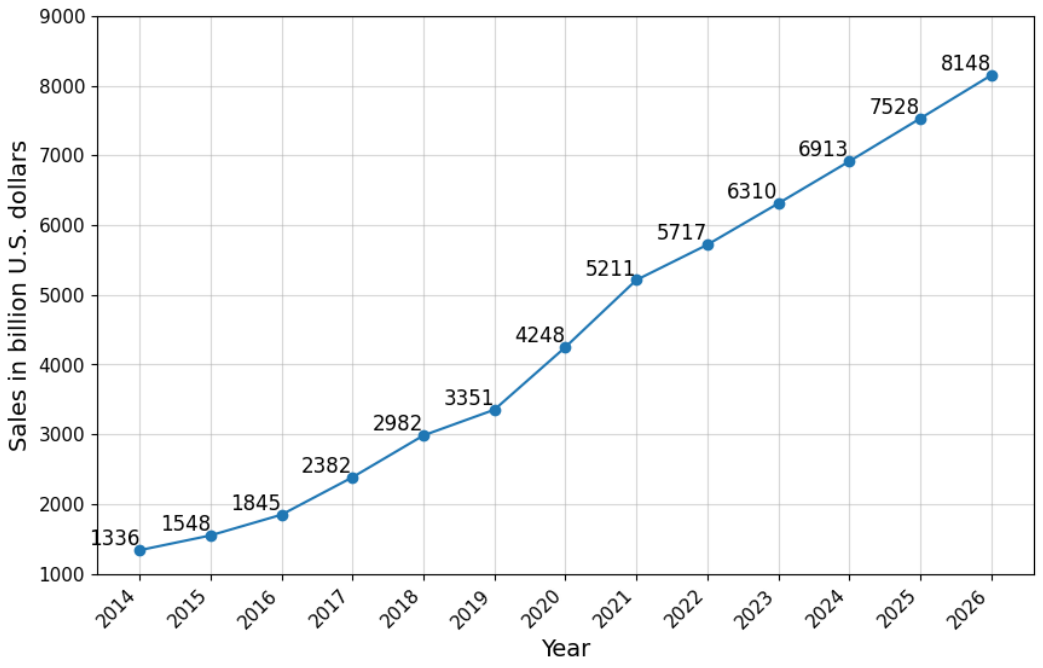


Chart 1: Retail e-commerce sales worldwide from 2014 to 2026

1.1.2. Impact on logistics

The transformative impact of e-commerce extends far beyond the realm of sales figures. Internet purchases and customized deliveries are the greatest trends impacting urban freight transport systems (Viu-Roig and Alvarez-Palau, 2020). Two primary factors drive this impact: the escalating volumes of goods being transported and the ever-increasing expectations of customers regarding the quality of service: in today's landscape, delivering orders to the doorstep has become the bare minimum. Customers consistently expect more (Bauer et al., 2020). It is important to note that last-mile delivery, which constitutes the final leg of the supply chain, stands out as the most challenging and least efficient part of the entire delivery process (Mangiaracina et al., 2019) and it is also the most cost-intensive phase (Van Heeswijk *et al*., 2020). This inefficiency in the last mile poses a considerable challenge for the logistics industry, demanding innovative and cost-effective solutions to meet the evolving demands of consumers.

1.2. Situation in Italy

With a more focused look at the e-commerce situation in Italy, it is evident that the e-commerce sector is experiencing consistent growth. Projections indicate an 11.59% increase in e-commerce from 2024 to 2027. This growth rate surpasses the global average (11.16%) and notably exceeds that of countries such as the United States (11.22%) (Statista, 2023). This trend highlights the fact that Italy is not only following the global trend but is even outpacing key nations in e-commerce growth.

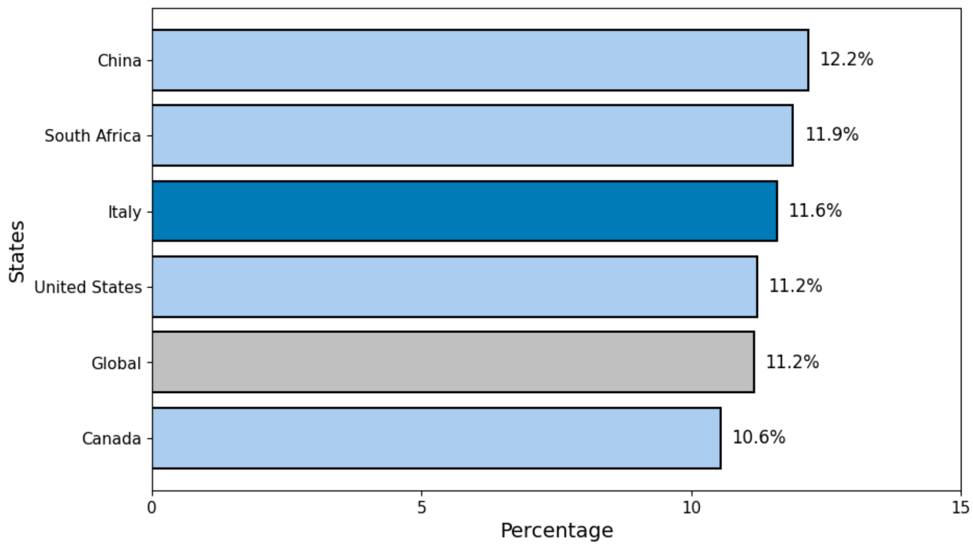


Chart 2: Retail e-commerce sales CAGR from 2023 to 2027, by country

One contributing factor to this sustained growth is the lasting impact of the COVID-19 pandemic. During Italy's quarantine measures, e-commerce played a vital role as people increasingly turned to online shopping for their essential needs. Importantly, this shift in consumer behavior has persisted even after the quarantine, establishing a new norm in Italian shopping habits.

Looking at the logistics of package delivery in Italy, this sector is dominated by seven key players who collectively control the entire market: BRT, SDA, GLS, Amazon Logistics, UPS, TNT-FedEx, DHL (AGCOM, 2023).

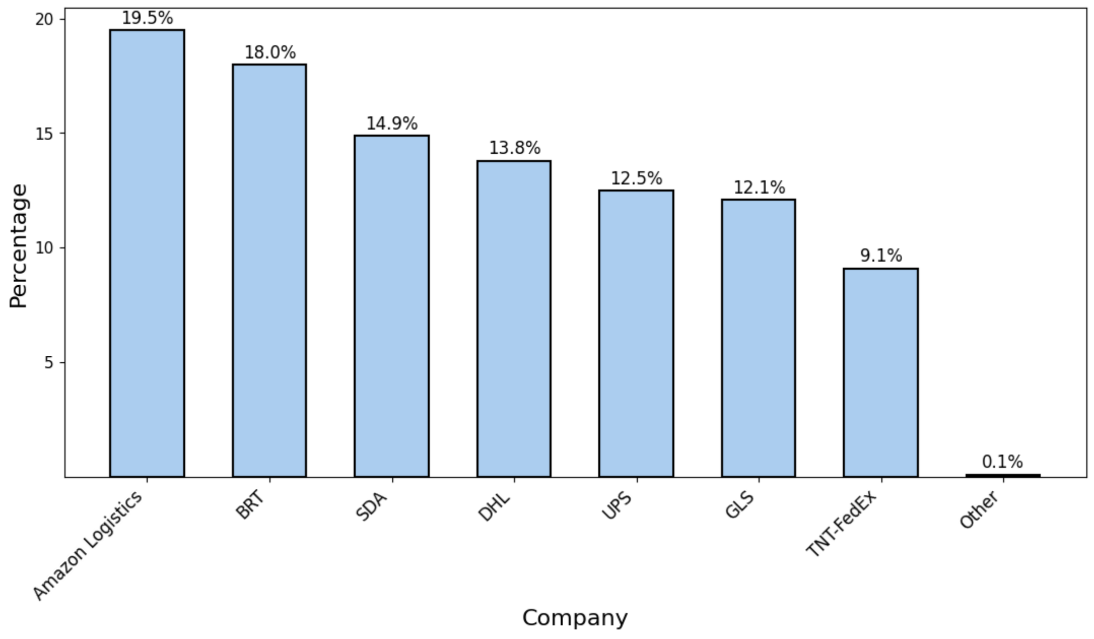


Chart 3: Competitive Framework

In light of the growing e-commerce trend in Italy, these companies have witnessed a significant upsurge in the volumes they must handle. From 2019 to 2023, there has been a remarkable 79.1% increase in domestic parcel delivery services alone, reflecting the expanding e-commerce landscape in the country (AGCOM 2023). Moreover, the surge in e-commerce has not been confined to domestic markets alone; international delivery services have also experienced substantial growth, with a notable increase of 58.3% during the same period.

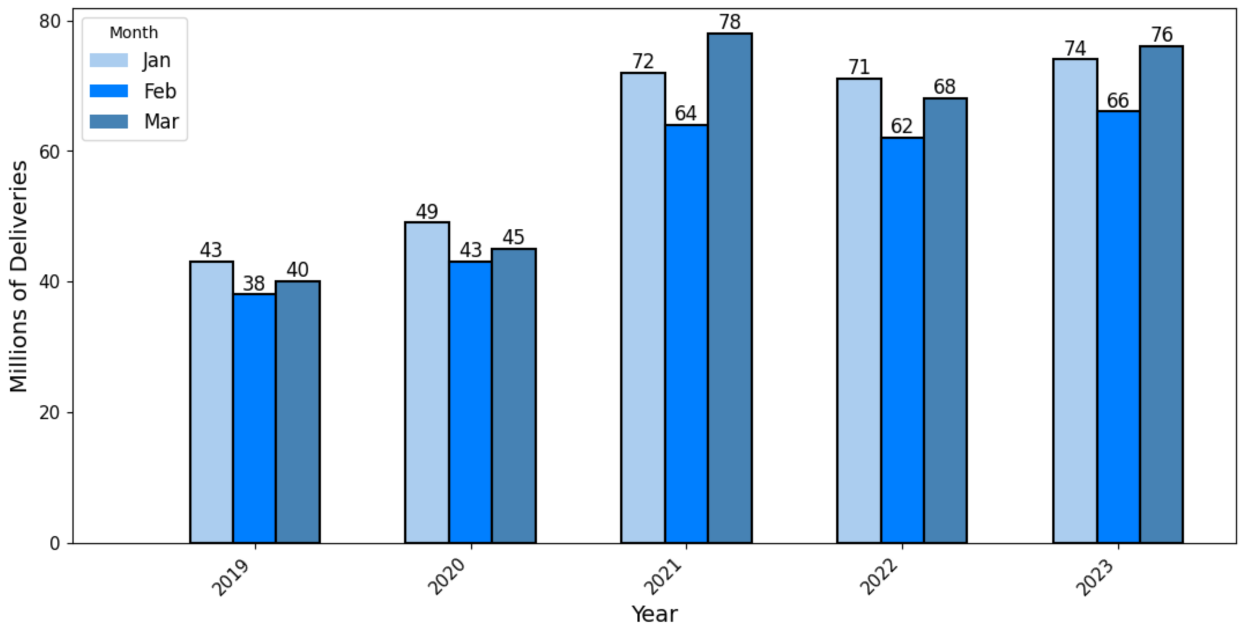


Chart 4: Parcel Delivery Service Volumes (National)

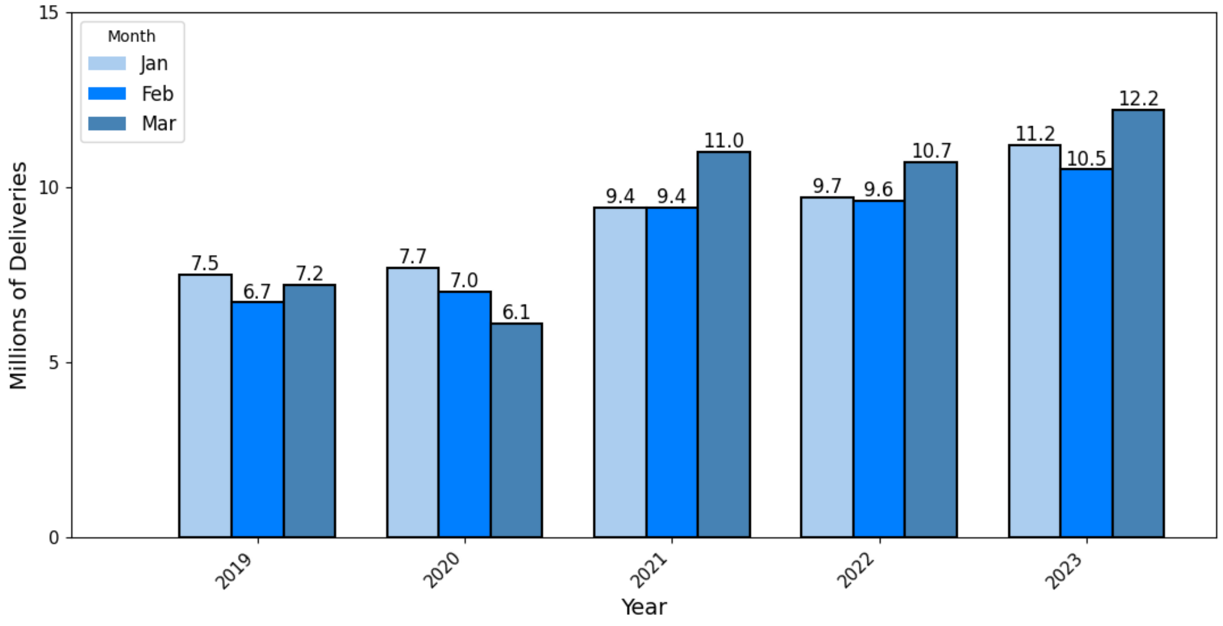


Chart 5: Parcel Delivery Service Volumes (International)

This surge in parcel delivery volumes poses substantial challenges for the Italian logistics industry, especially concerning last-mile deliveries. The inefficiencies and increased costs associated with this phase highlight the pressing need for more efficient and cost-effective solutions in the Italian market.

2 Literature review

As stated by Sun *et al*. (2007), " A distribution center that operates routing and scheduling scientifically could […] decrease travel distance and travel times […] and increase customer satisfaction”. Consequently, various models have been proposed and examined for simulating, studying, and optimizing delivery operations, particularly focusing on last-mile delivery.

The Vehicle Routing Problem (VRP) serves as the foundation for all the models developed in recent years. As articulated by Sopha *et al.* (2016), " VRP consists in determining a set of routes for vehicles based at a depot such that each customer is visited exactly once while minimizing overall transportation costs."

Numerous variations of the classic VRP exist, and Braekers *et al.* (2016) compiled a list of the most prominent ones:

1) The Capacitated Vehicle Routing Problem (CVRP) involves a single central depot, with each vehicle assigned to a unique route that shares identical characteristics.
2) The Vehicle Routing Problem with Pick Up and Delivery (VRPPD) requires a single vehicle to both collect goods from a specified location and deliver them to their destination, all within a single route.
3) The Vehicle Routing Problem with Backhauls (VRPB) entails vehicles making deliveries and pick-ups in a single route, accommodating the distinct needs of different customers.
4) The Multi-Depot Vehicle Routing Problem (MVRP) involves multiple depots scattered among the customer locations.

Expanding upon this foundation, recent research efforts have introduced innovative models to address evolving challenges in the field of the delivery process.

For instance, Yuyang Tan *et al.* (2019) have developed a CPRPPD (Capacitated Pollution Routing Problem with Pickup and Delivery) model for optimizing last-mile delivery that considers fuel consumption and greenhouse gas emissions and two practical delivery service options: home delivery (HD) and pickup site service (PS). To solve this problem, Tan's research proposes a two-phase heuristic algorithm that combines a hybrid ant colony optimization (HACO) and a multiple population genetic algorithm (MPGA) in the second stage.

Another important model is the S-MILE project proposed by Staniek and Sierpinski (2016): it is a smart platform that aims to support freight transport companies in planning, organizing, and realizing both the first and last mile in a more efficient way taking into account the particularities of environmental impact, the possibility of using EVs or other new technologies for freight transport.

In addition to these models, "An effective local search approach for the Vehicle Routing Problem with Backhauls" by Zachariadis and Kiranoudis (2012) proposes a new local search algorithm for the Vehicle Routing Problem with Backhauls (VRPB). The VRPB is a challenging combinatorial optimization problem that involves routing a fleet of vehicles to serve two types of customers: linehaul customers and backhaul customers. The objective is, of course, to minimize the total routing costs.

All these models and algorithms certainly offer excellent performance, but they are not capable of simulating and modeling all three phases of the delivery process together. Furthermore, they are quite complex and not user-friendly for the average user. They require a solid foundation and knowledge of programming languages and optimization algorithms. Moreover, making modifications to tailor them to specific needs can be challenging, limiting their flexibility.

Finally, there are no models that are specific to a particular area and rely on continually updated databases regarding the facilities of major couriers, to provide the most realistic simulation and modeling possible.

Therefore, the development of specific algorithms was necessary to address these limitations and provide a more comprehensive solution for the optimization of the delivery process. The model aims to bridge these gaps by offering a novel approach that encompasses all three phases of the delivery process, while also focusing on user-friendliness and adaptability.

3 Objectives and Methodology

3.1. Objectives

The primary objective of this project was to address a gap identified in the existing literature. Specifically, it aimed to create a program capable not only of modeling and simulating a part of the delivery process, such as the last-mile delivery, but also to extend this capability to cover all three major phases: pick-up, first-mile, and last-mile.

Another key goal was to establish a comprehensive database containing up-to-date information about the facilities of major Italian courier companies. This database served as a solid foundation for the subsequent program development, ensuring a simulation that closely mirrors reality.

Throughout the project, a pivotal consideration was to design a program that is as user-friendly as possible. This approach was adopted to ensure that the tool could be utilized by a diverse range of users, spanning from students and university research teams to retail store owners seeking to simulate or compare their shipping processes.

3.2. Methodology's structure

During the course of this project, a concerted effort was made to follow a linear path, focusing on one step at a time to achieve the utmost from each phase. The primary stages comprising the methodology's structure are illustrated in the figure below. This approach allowed for a systematic and methodical progression, optimizing the outcomes at each juncture.

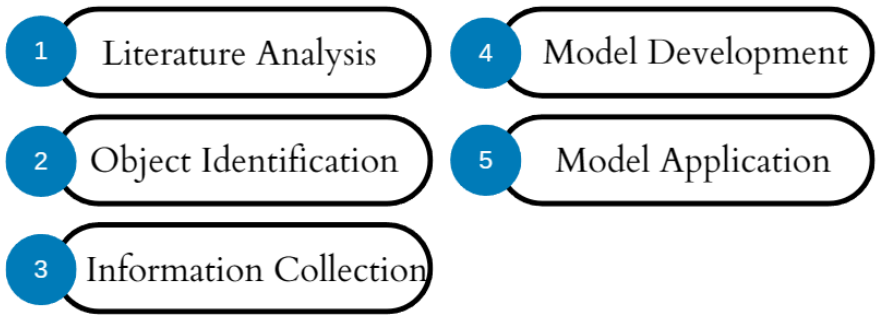


Figure 1: Methodology's structure

3.2.1. Literature analysis

First and foremost, a comprehensive literature review was conducted in search of models and algorithms for the modeling, simulation, and optimization of the entire delivery process, with a potential focus on couriers. This comprehensive review aimed to uncover pertinent themes, emerging trends, and methodologies while identifying gaps and potential avenues for future research.

3.2.2. Objective identification

As a result of the literature analysis conducted, gaps in the existing research were identified. Specifically, a comprehensive model that encompasses all stages of the delivery process with a specific state focus was not found. Moreover, the literature review revealed that the available models are often complex and challenging for the average user. Therefore, an additional objective was to design the program to be as user-friendly and flexible as possible, while preserving its inherent capabilities.

Additionally, it was observed that creating an updated database containing comprehensive information about the facilities of major courier services could significantly enhance the model's performance.

3.2.3. Information collection

Subsequently, the process entailed gathering information relating to the structure of the main Italian courier companies. Of particular interest was understanding the nature of these facilities, establishing their quantity within each company, and, most importantly, determining their geographic distribution across Italy.

The subsequent step involved organizing these newly acquired details into Excel spreadsheets, enabling the creation of structured datasets. These structured datasets facilitated in-depth analyses and served as a foundational resource for subsequent phases of the project. This systematic approach streamlined the data management process, ensuring its readiness for further utilization.

3.2.4. Model development

Once the database with all the necessary information was completed, the actual development of the model started. To achieve the objectives outlined in the "Objective Identification" section, Python was chosen as the programming language due to its numerous advantages in code development. Subsequently, three core programs were written, each addressing the first mile, last mile delivery, and pick-up phases, utilizing the previously created database.

For ease of use, Google Colab was selected as the programming environment. In addition to the standard Python libraries, a key external tool employed was OSRM, an open-source software that provides real-time duration, distance, and route information between two geographical points. This software played a pivotal role in enhancing the functionality and efficiency of the model.

3.2.5. Model application

Subsequently, the model underwent testing to evaluate its performance and assess how accurately it mirrored real-world scenarios. The testing phase aimed not only to gauge the model's accuracy but also to identify potential areas for improvement and fine-tuning. This thorough testing process ensured that the model's real-world applicability and reliability were carefully scrutinized.

4 Database creation

In this chapter, a more in-depth description of the various steps that led to the creation of the database subsequently employed in the program will be provided.

4.1. Facilities information retrieval

For the comprehensive facility assessment of the primary courier company operating in Italy, a multifaceted research approach was undertaken. Initial data collection involved an extensive search on the respective companies' websites to gather information related to the number, types, and locations of their facilities. Unfortunately, this proved to be a challenge, as only two companies, BRT and TNT-FedEx, had readily accessible information about their facilities on their official websites.

The next phase of data collection involved a thorough web search to identify any additional information from sources such as newspaper articles, scientific publications, or any publicly available data pertaining to the missing companies. In the case of Amazon Logistics, valuable insights were obtained from a Canadian logistics consulting company's website, which had conducted research and published an article detailing Amazon's global fulfillment center network. This article offered comprehensive and up-to-date information on Amazon's various facilities within Italy, with data current up to the first quarter of 2023.

Regarding DHL, primary information was sourced from a document available on the ACI website, dated September 2017, containing a list of DHL Express branches in Italy. Recognizing the potential limitations of this document, additional steps were taken to verify and update the data. This involved utilizing DHL's "Trova una Filiale" tool, accessible on their official website, to cross-reference and validate the information. Through this process, identification and exclusion of branches that had become inactive over the past six years, while incorporating newly opened branches into the dataset, ensured its accuracy and currency.

In the case of GLS and SDA, online sources provided only limited information, primarily encompassing the number of subsidiaries and the types of branches they maintain. To obtain more detailed insights, attempts were made to contact the customer service departments of both companies via email. Unfortunately, these efforts did not yield substantive information, as the responses received indicated that such data was not publicly available. Consequently, an alternative approach was pursued by leveraging the "Trova una Filiale vicino a me" tool offered on both companies' websites. This process, though time-consuming, required manual entry of numerous Italian municipalities to identify the served branches, ultimately resulting in the comprehensive mapping of Italy's coverage for these courier companies.

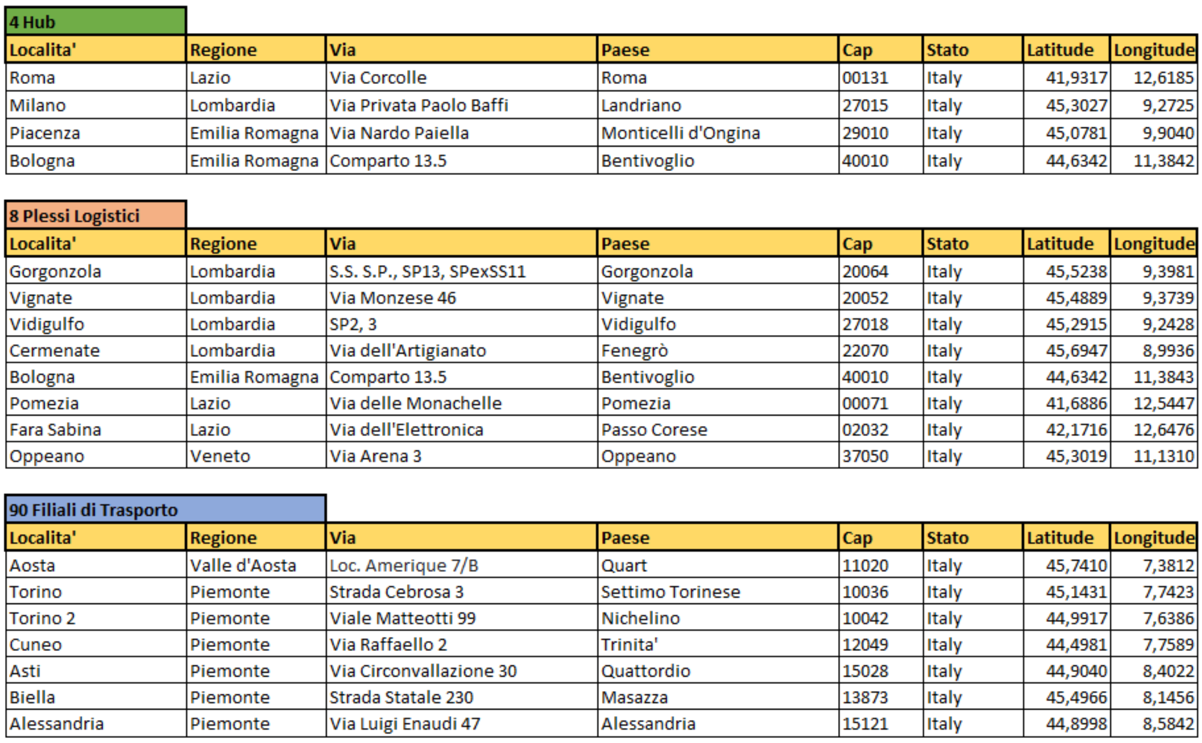
However, despite exhaustive efforts and the utilization of various research methods, it is regrettable to note that no information regarding UPS facilities was obtainable. This absence of data, despite thorough attempts, necessitates the omission of UPS from further consideration in the subsequent sections of this thesis.

All comprehensive information on the branches of the respective courier companies was successfully gathered, organized, and filtered, making it ready for utilization in the development of the database.

4.2. Database creation

After gathering all the necessary information, the database creation process commenced. Microsoft Excel was utilized for this purpose. The facilities of each company were categorized into Classes, based on their respective types, encompassing classic Branches, Hubs, or Logistics Complexes. Subsequently, the selection of parameters to be incorporated into the databases was determined. Ultimately, factors such as Location, Street, Country, Region, Postal Code (CAP), and the State (which consistently remains Italy in this specific context) were chosen for inclusion. Additionally, columns for Latitude and Longitude were introduced, although they are currently pending acquisition but deemed indispensable for the project's ongoing phases.

After the database had been filled with all previously collected information, the process shifted towards entering the coordinates for each individual branch of every company. Initially, a search was conducted for a tool capable of directly converting addresses into coordinates. An Excel add-on called MapCite, available for a fee, was discovered, claiming to accomplish this task with a high degree of accuracy and in a remarkably short time frame. However, following the utilization of this add-on, it became evident that the majority of coordinates provided were either imprecise or, in some instances, entirely erroneous. Consequently, the decision was made to undertake a manual search and input process for coordinates, ensuring a notably high level of precision. This meticulous approach was anticipated to be invaluable for subsequent phases involving code development for achieving the most realistic possible simulations and modeling. During this manual process, Google Maps was employed extensively to assist in locating and verifying accurate coordinates.



4 Hub

Localita'	Regione	Via	Paese	Cap	Stato	Latitude	Longitude
Roma	Lazio	Via Corcolle	Roma	00131	Italy	41,9317	12,6185
Milano	Lombardia	Via Privata Paolo Baffi	Landriano	27015	Italy	45,3027	9,2725
Piacenza	Emilia Romagna	Via Nardo Paiella	Monticelli d'Ongina	29010	Italy	45,0781	9,9040
Bologna	Emilia Romagna	Comparto 13.5	Bentivoglio	40010	Italy	44,6342	11,3842

8 Plessi Logistici

Localita'	Regione	Via	Paese	Cap	Stato	Latitude	Longitude
Gorgonzola	Lombardia	S.S. S.P., SP13, SPexSS11	Gorgonzola	20064	Italy	45,5238	9,3981
Vignate	Lombardia	Via Monzese 46	Vignate	20052	Italy	45,4889	9,3739
Vidigulfo	Lombardia	SP2, 3	Vidigulfo	27018	Italy	45,2915	9,2428
Cermenate	Lombardia	Via dell'Artigianato	Fenegrò	22070	Italy	45,6947	8,9936
Bologna	Emilia Romagna	Comparto 13.5	Bentivoglio	40010	Italy	44,6342	11,3843
Pomezia	Lazio	Via delle Monachelle	Pomezia	00071	Italy	41,6886	12,5447
Fara Sabina	Lazio	Via dell'Elettronica	Passo Corese	02032	Italy	42,1716	12,6476
Oppeano	Veneto	Via Arena 3	Oppeano	37050	Italy	45,3019	11,1310

90 Filiali di Trasporto

Localita'	Regione	Via	Paese	Cap	Stato	Latitude	Longitude
Aosta	Valle d'Aosta	Loc. Amerique 7/B	Quart	11020	Italy	45,7410	7,3812
Torino	Piemonte	Strada Cebrosa 3	Settimo Torinese	10036	Italy	45,1431	7,7423
Torino 2	Piemonte	Viale Matteotti 99	Nichelino	10042	Italy	44,9917	7,6386
Cuneo	Piemonte	Via Raffaello 2	Trinita'	12049	Italy	44,4981	7,7589
Asti	Piemonte	Via Circonvallazione 30	Quattordio	15028	Italy	44,9040	8,4022
Biella	Piemonte	Strada Statale 230	Masazza	13873	Italy	45,4966	8,1456
Alessandria	Piemonte	Via Luigi Enaudi 47	Alessandria	15121	Italy	44,8998	8,5842

Figure 2: Example of database [SDA]

4.3. Maps creation

After completing the database entry with all the necessary information, the process transitioned to the creation of an interactive HTML map for each company, showcasing the spatial distribution of branches throughout the Italian territory. Python was employed to craft these dynamic maps, allowing for enhanced interactivity and real-time updates. Different colors were strategically assigned to distinguish between Branches and Hubs, facilitating visual comprehension. Furthermore, each marker on the map featured a pop-up that appeared upon hovering, displaying essential details such as the branch's name and code (if applicable). This interactive approach not only enhanced the visual representation but also facilitated a user-friendly exploration of the company's facility network.

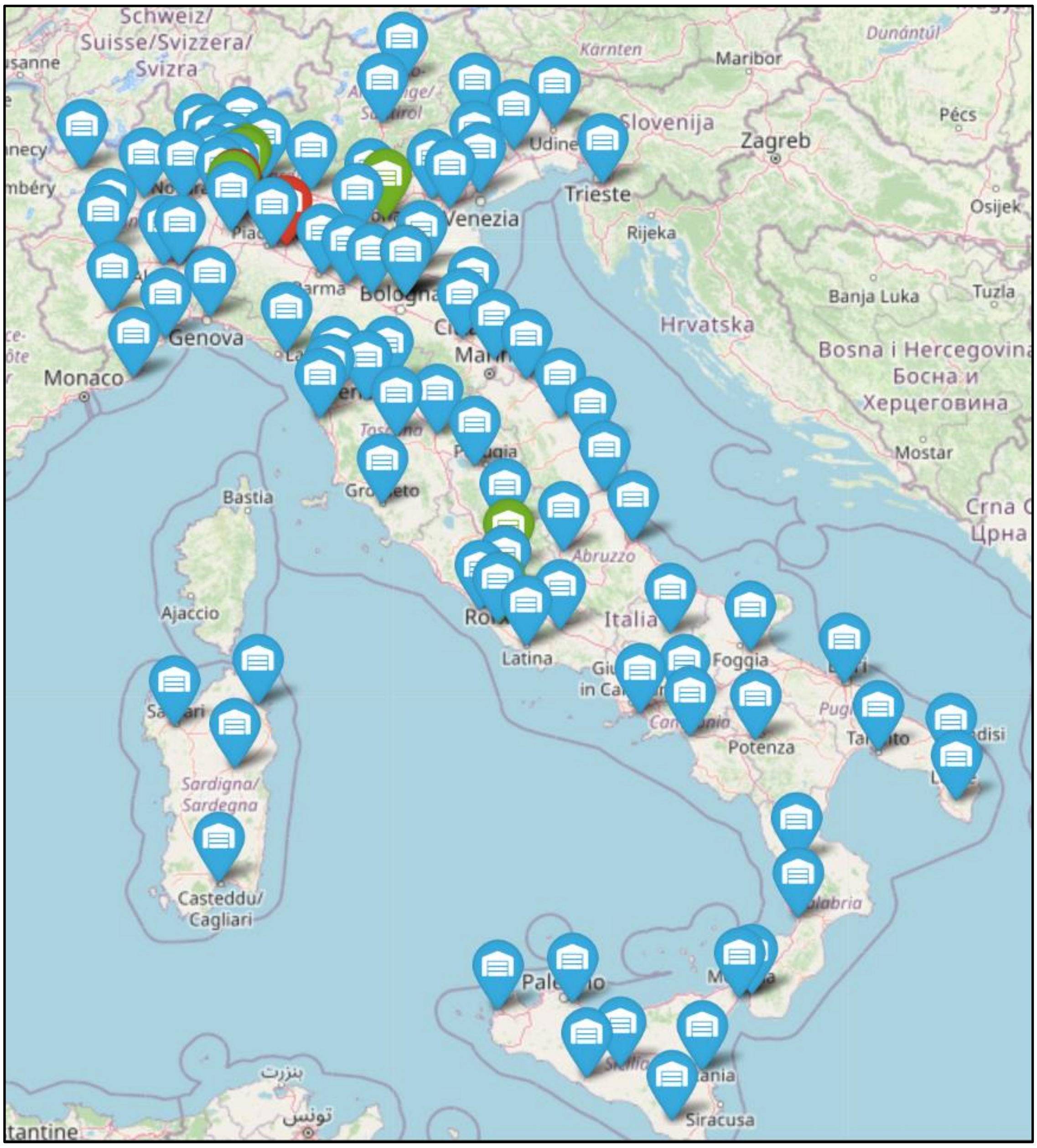


Figure 3: Example of Map [SDA]

5 Model

This chapter deals with the development of the simulation model. In particular, the three main code components are comprehensively explained (First Mile, Last Mile, and Pick-up), from their underlying logic to their output, with a focus on the most significant code segments.

5.1. First mile

The first code created and utilizing the previously established database focuses on the First Mile. The underlying concept of this code is to simulate a shipment from an origin point A to a destination point B. The code was developed and created using Python as the programming language and Google Colab as the platform.

5.1.1. Code explanation

The following flowchart provides a simplified representation of the program's operation, which will be explained in greater detail later.

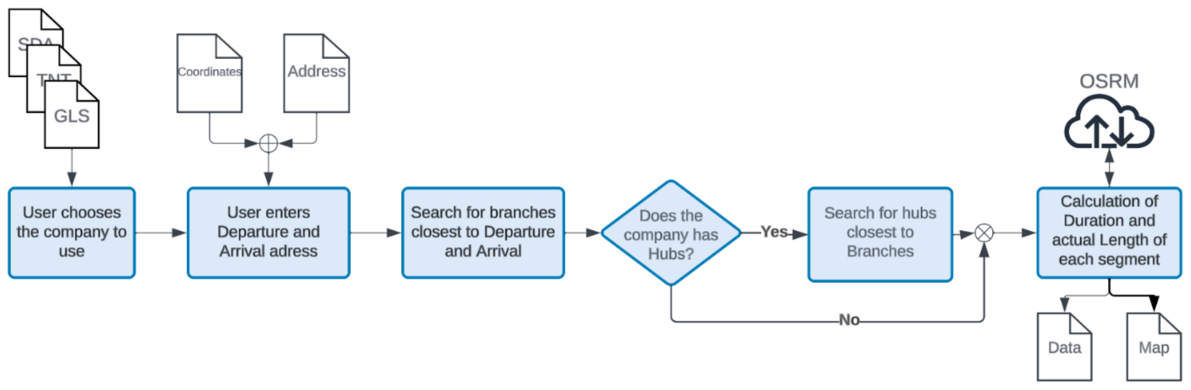

SDA
TNT
GLS
Coordinates
Address
OSRM
User chooses the company to use
User enters Departure and Arrival adress
Search for branches closest to Departure and Arrival
Does the company has Hubs?
Yes
No
Search for hubs closest to Branches
Calculation of Duration and actual Length of each segment
Data
Map

Figure 4: First-mile Code Flowchart

The program begins by prompting the end user to choose which courier company they want to use for the simulation. Then, the database of the selected company, containing all the facility information, is loaded into the program as input.

Subsequently, the user is also asked how they prefer to enter the starting and ending points: they can either enter precise coordinates directly or provide a complete address (Street, Street Number, City, ZIP Code, Province). For converting addresses to coordinates, the program utilizes Nominatim, a geocoding service that translates addresses into geographical coordinates. This feature was added to ensure a much simpler and more straightforward user experience.

After this, always using the provided database, the program calculates the distance between the two entered points and all the branches of the selected courier company. Consequently, two branches are identified and selected: the one nearest to the starting point and the one nearest to the destination point.

Once the two starting and destination branches are saved, there is a branching point in the program. It distinguishes between scenarios where the company has Hubs and/or Sorting Centers, implying a multi-level distribution network, or when the company only has branches.

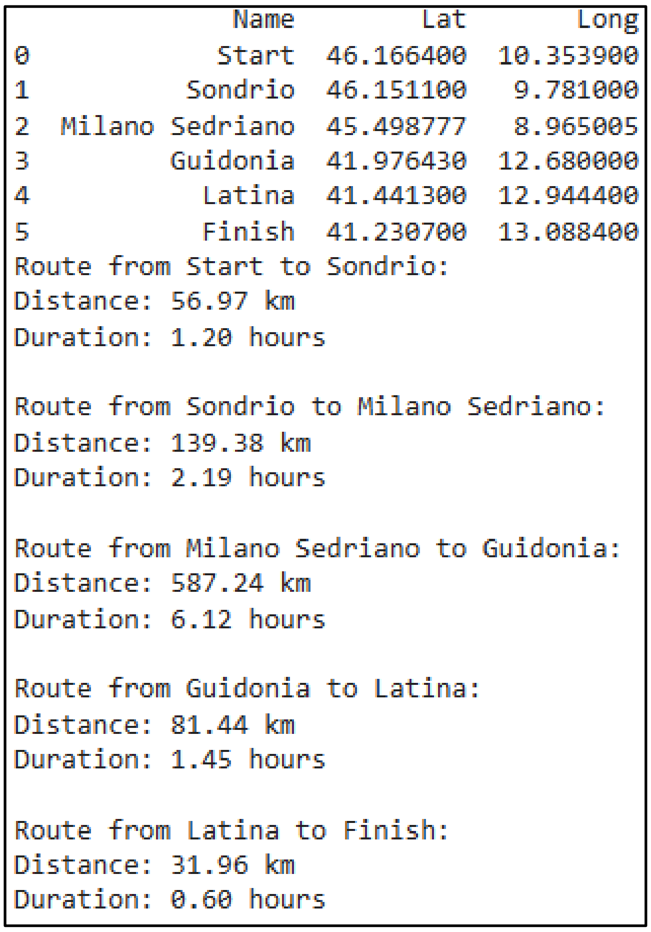
In cases where the company also has Hubs, the program, as done previously, calculates the distances between the two selected branches and all available hubs, identifying the two closest hubs to the starting and destination branches. In this case, six points are saved in memory: Starting Point, Branch Nearest to Starting Point, Hub Nearest to Branch 1, Hub Nearest to Branch 2, Branch Nearest to Destination, and Destination Point. Conversely, if the initially selected company lacks Hubs, the program skips this step, resulting in the storage of only four points: Starting Point, Branch Nearest to Starting Point, Branch Nearest to Destination, and Destination Point.

At this point, the program has stored 6 (or 4 in the 'Only Branches case') points in memory. It then utilizes OSRM, an open-source routing tool, to calculate the actual distance and real-time duration for these 5 (or 3) segments.

In the end, the program displays the calculated distances and durations on the screen. Additionally, it generates a map that visualizes the actual route taken by the hypothetical package under consideration.

5.1.2. Code output

Now, an example output obtained from a test will be presented, with Sonico in Val Camonica selected as the starting point, Rieti in Lazio as the destination, and BRT was chosen as the courier company.



```
              Name        Lat       Long
0            Start  46.166400  10.353900
1          Sondrio  46.151100   9.781000
2  Milano Sedriano  45.498777   8.965005
3         Guidonia  41.976430  12.680000
4           Latina  41.441300  12.944400
5           Finish  41.230700  13.088400
Route from Start to Sondrio:
Distance: 56.97 km
Duration: 1.20 hours

Route from Sondrio to Milano Sedriano:
Distance: 139.38 km
Duration: 2.19 hours

Route from Milano Sedriano to Guidonia:
Distance: 587.24 km
Duration: 6.12 hours

Route from Guidonia to Latina:
Distance: 81.44 km
Duration: 1.45 hours

Route from Latina to Finish:
Distance: 31.96 km
Duration: 0.60 hours
```

Figure 5: First-mile code data output

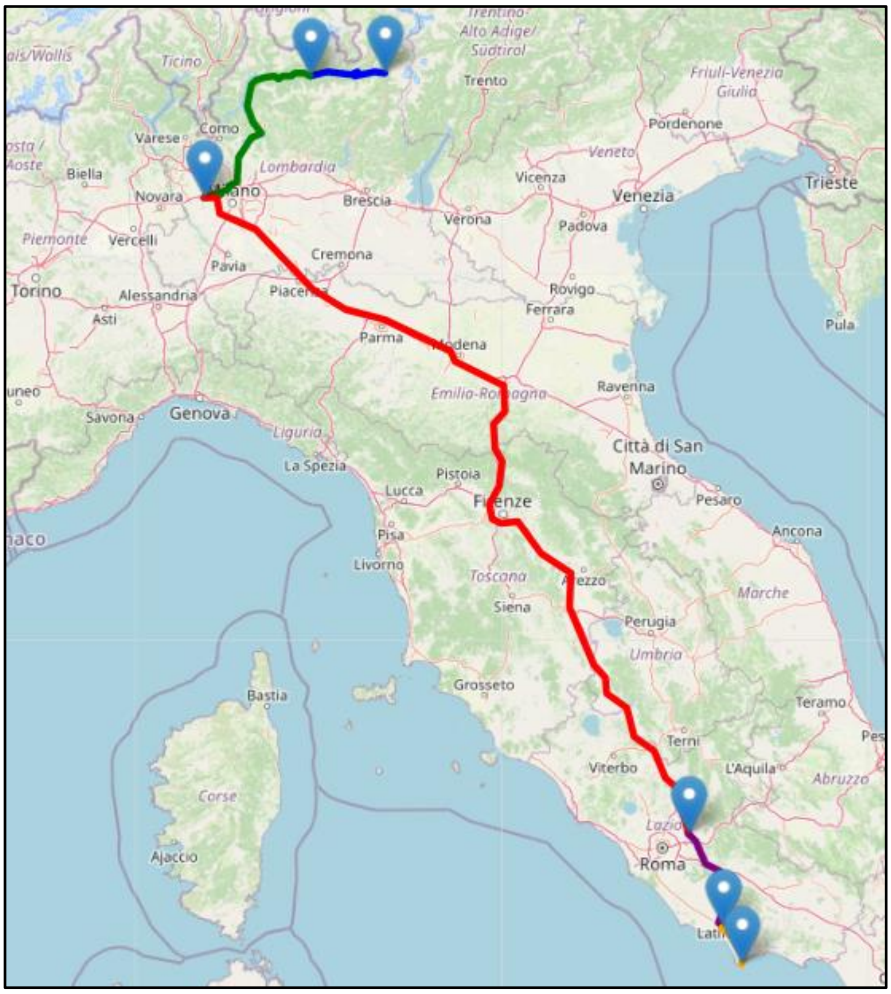


Figure 6: First-mile code map output

From the output, it can be noted how the code operates: it displays the two selected branches (Sondrio and Latina in this case) and the two hubs (Milano Sedriano and Guidonia). Additionally, real durations and distances for each segment are shown. Lastly, a map is generated to provide a higher-level overview of the entire route and the various stops.

5.1.3. Code in detail

In this section, a detailed analysis of the program's key code lines will be conducted to grasp the code's operations without delving excessively into specific details, considering the code's considerable length.

```
def geocode_address(address):
    nominatim_url =
f"https://nominatim.openstreetmap.org/search?format=json&q={address}"
    response = requests.get(nominatim_url)
    data = response.json()
    if len(data) > 0:
        latitude, longitude = float(data[0]['lat']),
float(data[0]['lon'])
        return latitude, longitude
    else:
        print("Address not found.")
        return None, None
```

These lines define a function called `geocode_address` that takes an address as input and uses the Nominatim service from OpenStreetMap to convert it into latitude and longitude coordinates. If a valid result is found, it returns the coordinates; otherwise, it returns `None`. It is used at the start of the code when the user inserts the starting and ending points.

```
df['Distance_Start'] = df.apply(
        lambda row: geodesic((row['Latitude'], row['Longitude']),
(start_latitude, start_longitude)).km, axis=1)

    df['Distance_Finish'] = df.apply(
        lambda row: geodesic((row['Latitude'], row['Longitude']),
(finish_latitude, finish_longitude)).km, axis=1)

    # Filter the DataFrame to get rows with 'Classe' == 'B'
    df_filiali = df[df['Classe'] == 'B']

    # Find the closest point to Start and Finish
    closest_start = df_filiali.loc[df_filiali['Distance_Start'].idxmin()]
    closest_finish=df_filiali.loc[df_filiali['Distance_Finish'].idxmin()]
```

In these lines, the distances between the two previously input points and all branches of the company in the database are calculated. Note that the distances here are calculated using the Haversine formula, which is commonly used to calculate distances between two points on the surface of a sphere, such as the Earth, and not the effective road distance. This choice will be discussed later, in the program’s considerations and conclusions. Subsequently, the program filters the DataFrame to include only the branches (Classe = B) preparing for the selection of the two branches nearest to the input points and leaving out the hubs (Classe = A) for now.

```
df_hubs = df[df['Classe'] == 'A']

df_hubs['Distance_Start'] = df_hubs.apply(lambda row: geodesic
((closest_start['Latitude'], closest_start ['Longitude']),
(row['Latitude'], row['Longitude'])).km,axis=1)


df_hubs['Distance_Finish'] = df_hubs.apply(lambda row: geodesic
((closest_finish['Latitude'], closest_finish['Longitude']),
(row['Latitude'], row['Longitude'])).km, axis=1)

closest_hub_start = df_hubs.loc[df_hubs['Distance_Start'].idxmin()]
closest_hub_finish = df_hubs.loc[df_hubs['Distance_Finish'].idxmin()]
```

In this section of the code, a similar process is followed as before, with a slight difference. Firstly, a sub-data frame is created, containing only the hubs. Then, the code calculates the distances between each hub and the two previously selected branches, ultimately selecting the two hubs that are closest in proximity. Similar to previous calculations, the Haversine formula is employed to compute these distances.

```
for i in range(len(location_pairs)):
    start, end = location_pairs[i]

    start_coords = points_df.loc[points_df['Name'] == start,['Lat',
'Long']].values[0]
    end_coords = points_df.loc[points_df['Name'] == end, ['Lat',
'Long']].values[0]

    request_url = f"http://router.project-osrm.org/route/v1/
car/{start_coords[1]},{start_coords[0]};{end_coords[1]},{end_coords[0]}?g
eometries=polyline&steps=true&annotations=true"

     r = requests.get(request_url)route_data= json.loads (r.content)
["routes"][0]
```

This section of code is the core component of the program, responsible for requesting route information from the OSRM routing service for various segments of the route. The code operates within a loop that iterates through a list called `location_pairs`, which contains the pairs of the points previously found (Starting and Ending points, the 2 branches, and the 2 hubs).

Overall, these four code sections are of paramount significance. The first segment defines a crucial function responsible for geocoding addresses into coordinates. The second and third sections are pivotal for calculating distances, aiding in the selection of the nearest branches and hubs. Lastly, the fourth part orchestrates requests to the OSRM software, facilitating the retrieval of driving distances and durations between the designated points.

5.1.4. Functionalities and Limitations

In summary, it can be stated that the program operates in a straightforward and intuitive manner, making it accessible to any user. This ease of use is attributed to the minimal interactions required, which have also been as fully streamlined as possible. These interactions include selecting the preferred company to use and deciding whether to input the complete address or directly input coordinates, thereby ensuring a smooth and user-friendly experience.

The program handles different scenarios in distinct ways. As mentioned earlier, the code functions differently when a company has both branches and hubs compared to when it only has branches.

Another feature incorporated into the program is that, in the rare event of a company that has hubs and the origin and destination branches being the same (a scenario with low but non-zero probability), the program recognizes this exception and behaves differently. In such a case, the package's route does not include a stop at the hub, as it would be an unnecessary detour, resulting in a loss of time and a waste of resources, since the package would need to return to the same branch.

Certainly, the program can be modified to accommodate numerous scenarios. For example, it can handle cases where passing through a hub significantly extends the distance and time compared to a direct connection between two branches. However, it should be noted that different companies often have their own distinct approaches to managing various cases and exceptions. Given that detailed knowledge of these company policies may become available in the future, it is worth mentioning that Python programming ensures that the program is highly scalable. This makes it straightforward to incorporate additional components as needed to align with the policies of various companies when such information becomes accessible.

The program's only limitation, if it can be referred to as such, arises during the selection of the two nearest branches to the starting and ending points, as well as the two Hubs. As previously explained, in this case, distance is not obtained through OSRM connection but is calculated using the Haversine formula. This implies that the obtained distance is not the actual distance but rather the "as-the-crow-flies" distance. In most cases, this is not an issue when selecting the nearest branches and hubs because actual distance and as-the-crow-flies distance are closely related. However, there can be situations, particularly in mountainous regions, where these two values can differ significantly due to the winding paths that roads take through the mountains. This can affect the selection, which may not be the most efficient. Therefore, it is advisable to consult the map provided along with the output to verify the accuracy of the selection.

One way to address this "limitation" is to calculate distances between points and branches using OSRM to obtain more precise values and ensure that the selected branches or hubs are always the closest. However, this would lead to a substantial increase in computational time, which currently stands at approximately 2 seconds. Therefore, resolving the issue in the very very few cases where it arises may not justify a significant increase in processing time.

5.2. Last mile

The second code instead focuses on the Last-Mile. The aim of the code is to simulate a last-mile delivery route while concurrently identifying the optimal path. As the previous code, it was developed and created using Python as the programming language and Google Colab as the platform.

5.2.1. Code Explanation

The following flowchart provides a simplified representation of the program's operation, which will be explained in greater detail later.

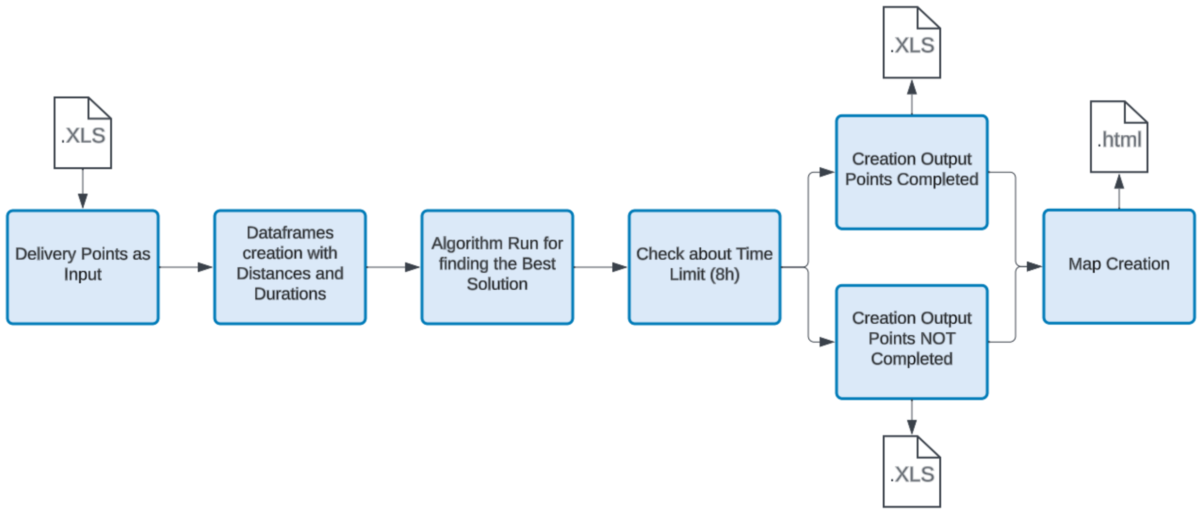

.XLS
Delivery Points as Input
Dataframes creation with Distances and Durations
Algorithm Run for finding the Best Solution
Check about Time Limit (8h)
.XLS
Creation Output Points Completed
Creation Output Points NOT Completed
.XLS
.html
Map Creation

Figure 7: Last-mile code flowchart

The program commences by taking as input in the form of an XSLX file containing coordinates for all the scheduled delivery points. Subsequently, the program prompts the user to select the company for simulation and, from there, the branch from which to initiate the delivery route. Once this information is acquired, the program establishes a connection with the OSRM software to create two new data frames: one for duration and one for distance, both encompassing all possible pairings between points. This process is time and resource-intensive due to the significantly high number of potential pairings.

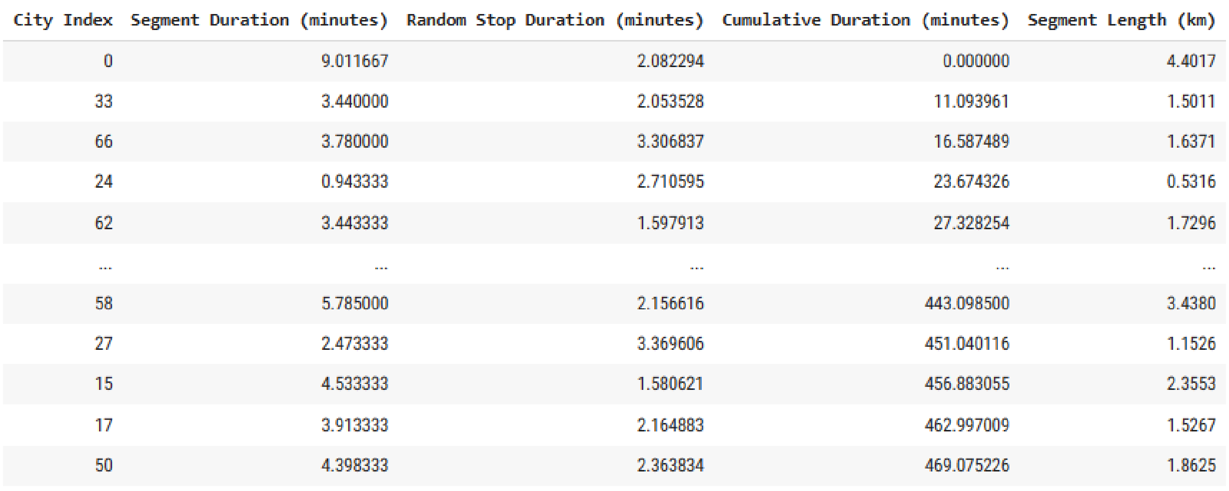
Subsequently, the data frame containing all the durations is handed over to the resolution algorithm, which will be elaborated upon in detail later. Once the program completes its computation, the resulting output is an array of indexes that corresponds to the optimal order of visitation for all stops. It should be noted that the stop duration for each stop is taken into account and will be explained in greater detail later on.

Therefore, after the check concerning the 8-hour work limit, the program generates two distinct outputs: the first one lists the completed stops, including the duration of each segment, stop duration, cumulative duration, and segment distance. The second output comprises a list of stops that couldn't be executed due to the elapsed time.

At the end, just like in the previous code, a map depicting the obtained route is also generated to provide a higher-level overview.

5.2.2. Code Output

Now, the output of a test conducted with 79 delivery points, selected in the eastern area of the city of Milan (Citta' Studi/Acquabella), will be displayed:



City Index	Segment Duration (minutes)	Random Stop Duration (minutes)	Cumulative Duration (minutes)	Segment Length (km)
0	9.011667	2.082294	0.000000	4.4017
33	3.440000	2.053528	11.093961	1.5011
66	3.780000	3.306837	16.587489	1.6371
24	0.943333	2.710595	23.674326	0.5316
62	3.443333	1.597913	27.328254	1.7296
...	...	...	...	...
58	5.785000	2.156616	443.098500	3.4380
27	2.473333	3.369606	451.040116	1.1526
15	4.533333	1.580621	456.883055	2.3553
17	3.913333	2.164883	462.997009	1.5267
50	4.398333	2.363834	469.075226	1.8625

Figure 8: Last-mile Code Data Output

As observed, the program outputs a table containing all the information related to the optimal route that has been identified. The 'City Index' column represents the optimal order for visiting the delivery points. The 'Segment Duration' and 'Segment Length' columns, in turn, indicate the duration and distance of each segment. For example, between point 0 and point 33, the duration is 9.01 minutes, and the distance is 4.40 km.

The 'Random Stop Duration' column lists the stop durations for each stop. Lastly, the 'Cumulative Duration' represents the total time elapsed, considering both the segment duration and stop duration. This column is used to verify compliance with the 8-hour working limit.

As previously explained, the program also generates a map depicting the entire route that the courier ideally follows from one stop to another during a working day.

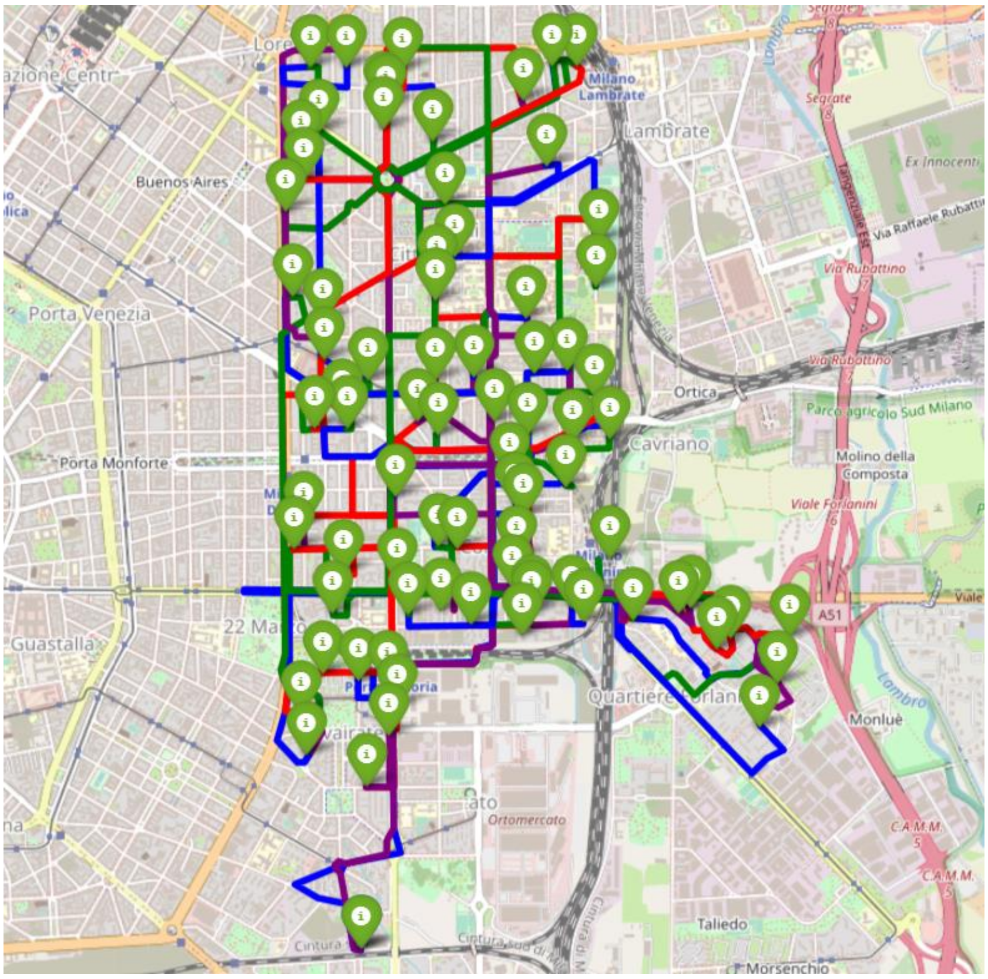


Figure 9: Last-mile Code Map Output

5.2.3. The algorithm

The algorithm employed in the code is designed to address the Traveling Salesman Problem (TSP), a well-known optimization challenge involving the determination of the shortest route that visits a set of points and returns to the starting point. This approach utilizes a genetic algorithm, a heuristic optimization technique inspired by the principles of natural selection, to derive approximate solutions for complex optimization and search problems. The fitness function, denoted as `fitness_dists`, plays a pivotal role in this process by assessing the quality of candidate solutions based on the duration matrix between addresses. Key parameters, including the mutation probability (`mutation_prob`) and the maximum number of optimization attempts (`max_attempts`), are configured to influence the algorithm's behavior.

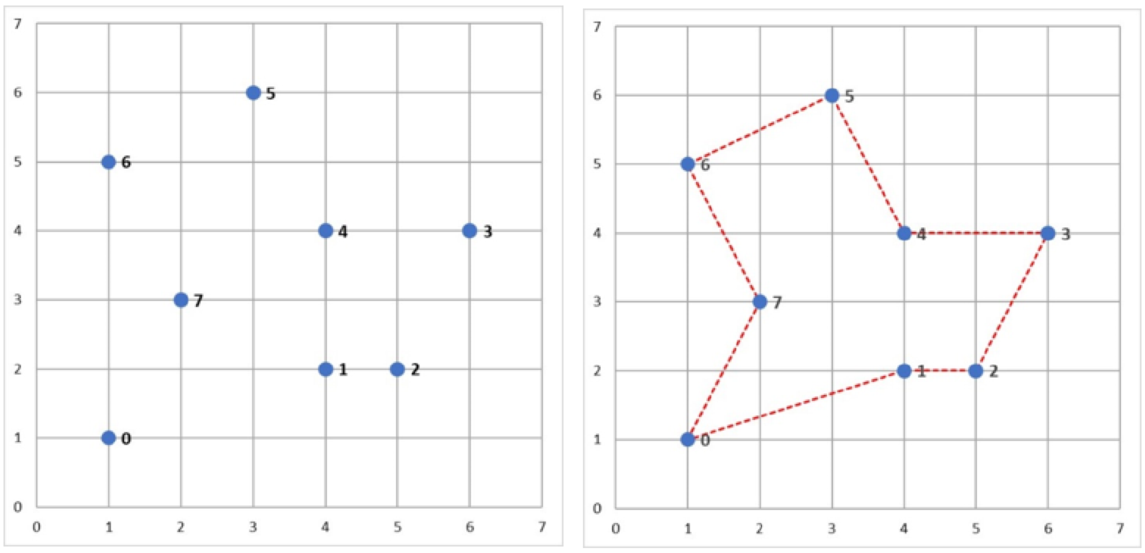


Figure 10: Traveling Salesman Problem Example

The implementation of this genetic algorithm leverages the mlrose library in Python. Mlrose, or Machine Learning Randomized Optimization and Search, is a versatile and powerful library specifically designed for solving complex combinatorial optimization problems using randomized algorithms. In this case, mlrose is instrumental as it abstracts many of the intricacies involved in algorithm development and optimization.

The outcome of this process comprises the `best_state`, represented by the optimal path for visiting the given addresses. This genetic algorithm approach, enhanced by the mlrose library, serves as a valuable tool for finding near-optimal solutions to the TSP, particularly when dealing with a substantial number of addresses where exhaustive search methods become computationally unfeasible. It streamlines the implementation of the genetic algorithm, allowing practitioners to focus on problem-solving and efficient solutions for large-scale address optimization challenges.

To assess the algorithm's performance and determine if the generated output is indeed optimal and provides a highly accurate representation of reality, a test was conducted. Given that, on average, a courier can visit approximately 80 locations within a city such as Milan in an 8-hour workday, a test was initiated using a database of 80 delivery points specifically designed for this purpose. The test results revealed that the program calculated a time of 476 minutes, which is remarkably close to the 8-hour workday benchmark. From this observation, which has been taken as an example and shown in the previous paragraph, it can be inferred that the algorithm employed is highly effective and adept at representing reality at an exceptional level.

5.2.4. Code in detail

In this section, a detailed analysis of the program's key code lines will be conducted to grasp the code's operations without delving excessively into specific details, considering the code's considerable length.

```
def get_distance(point1: dict, point2: dict) -> tuple:
    url = f"""http://router.project-osrm.org/route/v1/ driving/
{point1["lon"]},{point1["lat"]};{point2["lon"]},{point2["lat"]}?overview=
false&alternatives=false"""
    r = requests.get(url)
    route = json.loads(r.content)["routes"][0]
    return (route["distance"], route["duration"])
```

The `get_distance` function is designed to determine the distance and duration between two geographic points using the Open Source Routing Machine (OSRM) API. It takes two dictionaries, each representing a point with latitude and longitude coordinates.

'`overview=false`': this parameter tells the OSRM API not to provide a high-level overview of the route; it's set to "false" to get more detailed route information. '`alternatives=false`': This parameter instructs the API not to provide alternative routes. It's set to "false" to get only the primary route. The function returns the distance and duration as a tuple. It is one of the most important parts of the code since it is used to obtain distances and durations for every possible pair.

```
for i in range(len(points_df)):
    point1 = {"lat": points_df.loc[i, 'Lat'], "lon": points_df.loc[i,
'Long']}
    for j in range(i + 1, len(points_df)):
        point2 = {"lat": points_df.loc[j, 'Lat'], "lon": points_df.loc[j,
'Long']}
        distance, duration = get_distance(point1, point2)
        dist_array.append((points_df.loc[i, 'ID'], points_df.loc[j,
'ID'], duration))
        distance_array.append((points_df.loc[i, 'ID'], points_df.loc[j,
'ID'], distance))

duration_df = pd.DataFrame(dist_array, columns=["Origin", "Destination",
"Duration(s)"])
distance_df = pd.DataFrame(distance_array, columns=["Origin",
"Destination", "Distance(m)"])
```

This part of the code systematically calculates distances and durations between pairs of geographic points within the DataFrame `points_df`, which contains the list of all the delivery points and the selected branch. Using a nested loop structure, it ensures each unique pair of points is considered once to prevent redundant calculations. The `get_distance` function is applied to each pair to compute distance and duration, with the results stored in the `dist_array` and `distance_array`, along with the IDs of the respective points. As previously mentioned, this particular section of the code demands the most time and resources, primarily due to the high volume of pairs generated. The number of pairs can be calculated using the binomial coefficient: *#pairs* = $\binom{n+1}{2}$ with n+1 that stand for the number of delivery points + the branch.

At the end, it creates 2 new data frames, `duration_df` and `distance_df` from the arrays (`dista_array` and `distance_array`) that were created just before.

```
fitness_dists = mlrose.TravellingSales(distances=dist_array)
problem_fit = mlrose.TSPOpt(length=len(points_df),
fitness_fn=fitness_dists, maximize=False)

np.random.seed(2)

best_state, best_fitness = mlrose.genetic_alg(problem_fit,
mutation_prob=0.2, max_attempts=750, random_state=2)
```

This code segment serves as the central processing unit of the entire program. The algorithm, as previously elucidated, now takes as input the DataFrame containing duration data (the rationale behind choosing durations as the optimization parameter rather than distances will be explained later). To ensure reproducibility in the results, a random seed is set using `np.random.seed(2)`: this is particularly useful when conducting experiments or testing. The program, in turn, produces `best_state` as its output, representing the array with the optimal visiting order of all the delivery points.

```
cumulative_duration = 0
time_limit_exceeded = False

for i, (city_index, segment_duration) in enumerate(zip(best_state,
segment_durations_minutes)):
    random_duration = random.uniform(2, 4)
    random_stop_durations.append(random_duration)

    if cumulative_duration + segment_duration + random_duration <=
time_limit:
        completed_stops.append({'City Index': city_index, 'Segment
Duration (minutes)': segment_duration, 'Cumulative Duration (minutes)':
cumulative_duration})
        cumulative_duration += segment_duration + random_duration

     else:
        uncompleted_stops.append({'City Index': city_index, 'Segment
Duration (minutes)': segment_duration, 'Cumulative Duration (minutes)':
cumulative_duration})
        time_limit_exceeded = True
```

Here the code initializes `cumulative_duration` and a flag, `time_limit_exceeded`, to keep track of the total time spent and whether a time limit has been exceeded. It iterates through the list of stop indices and their respective segment durations. For each delivery point, a random stop duration (between 2 and 4 minutes) is generated and added to a list. The code then checks if adding the stop's segment duration and the random stop duration would keep the cumulative duration within the specified time limit (480 minutes or 8 hours). If it does, information about the completed stop is appended to one list (`completed_stops`), and `cumulative_duration` is updated. However, if adding the stop exceeds the time limit, the code appends information about the uncompleted stop to another list (`uncompleted_stops`) and sets the `time_limit_exceeded` flag to True.

In summary, these four code segments are the most important and necessary for understanding how the entire program functions. The first one defines a function that is later used to calculate the distance and duration for each pair of points. The second one utilizes this function to obtain distance and duration for every possible segment, creating two data frames where all the duration and distance values are stored. The third one is the core of the entire program: here is where the algorithm is launched to obtain the best possible sequence. Finally, the last one is responsible for generating a random stop duration for each individual stop and performing the check regarding the 8-hour working limit.

5.2.5. Functionalities and Limitations

In this section, the program's functionalities and limitations will be discussed. Let's begin by addressing why the algorithm optimizes based on duration rather than distance, as is typically expected for a Traveling Salesman Problem (TSP). The decision to consider duration is rooted in the need to account for the stop durations that must be observed at each stop to emulate real-world conditions. Additionally, the choice aligns with the practicality of limiting the total stoppage time within an 8-hour workday. Importantly, selecting duration over distance does not compromise the quality of the output, as real distance and duration data are sourced from the OSMR software.

Regarding stop durations, a random duration between 2 and 4 minutes was chosen to simulate the variability encountered in a courier's daily routine. Factors like the customer's pace in reaching the street, parking availability, and more influence stop durations.

To further enhance realism, an 8-hour work limit was introduced. When the cumulative duration surpasses this limit, the uncompleted stops are flagged and documented in a separate file, highlighted in red on the map. As previously mentioned, these adjustments aim to ensure the most realistic simulation possible.

To confirm the program's accuracy in achieving an efficient route within the 8-hour constraint, a test was conducted in the eastern area of Milan with 80 delivery points, which represents the daily average for a courier. The results were satisfactory, indicating that it would be possible to visit all points and return to the starting branch within 476 minutes, nearly exactly 8 hours. This outcome validates the program's correctness and its applicability for simulation and analysis.

As for limitations, two significant constraints exist: computational time and the inability to simulate under different conditions. Computational time is understandably substantial, with approximately 40 minutes for distance and duration calculations and around 40 minutes for the core algorithm. It's important to recognize the substantial volume of data the program handles—considering all possible segments, with 80 delivery points, results in 3160 requests to OSRM for segment distance and duration data. The same applies to the algorithm, which conducts numerous tests on a multitude of data to obtain the optimal solution. Therefore, the time required is justifiable and directly proportional to the volume of data processed.

It would have been highly interesting to conduct simulations using the program under different conditions, such as different time slots and days of the week. This would have allowed for an analysis of the courier's performance in scenarios with varying traffic conditions. Regrettably, this cannot be achieved with the current OSRM software, as it provides standardized distance and duration data, which are obtained as averages across a wide range of diverse situations. This includes peak traffic hours and less congested times of the day, as well as days with heavy traffic compared to those with lighter traffic.

5.3. Pick-up

The third and final code was developed to simulate the pick-up process carried out by the courier to collect packages from warehouses and stores. Like the previous codes, this one was also written in Python using Google Colab.

5.3.1. Code Explanation

The following flowchart provides a simplified representation of the program's operation, which will be explained in greater detail later.

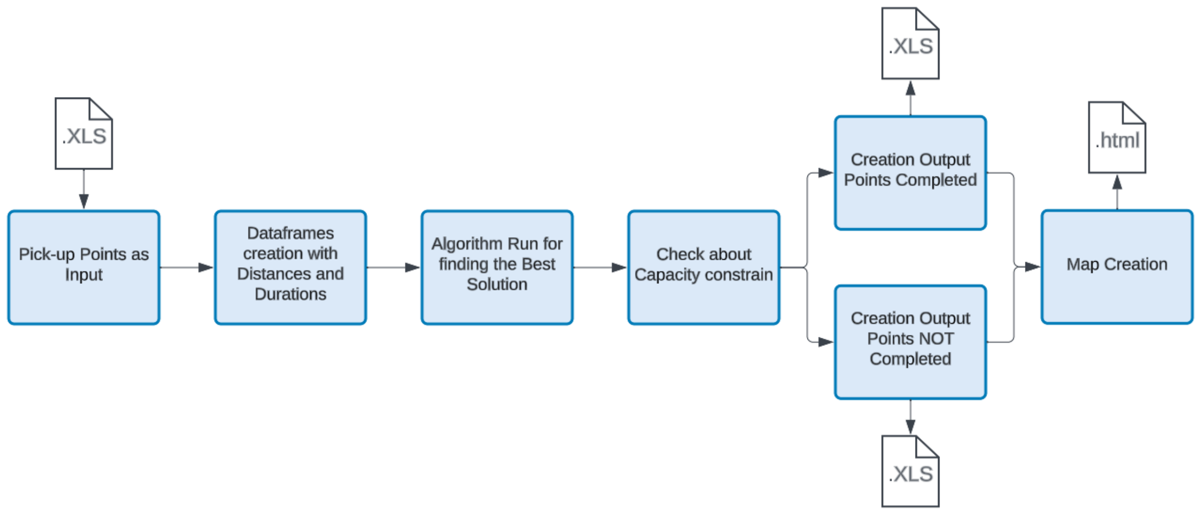

.XLS
Pick-up Points as Input
Dataframes creation with Distances and Durations
Algorithm Run for finding the Best Solution
Check about Capacity constrain
.XLS
Creation Output Points Completed
Creation Output Points NOT Completed
.XLS
.html
Map Creation

Figure 11: Pick-up Code Flowchart

The program operates in a manner similar to that of Last Mile delivery, with some adjustments to tailor it for the specific use case. The program begins by taking input in the form of a list containing the coordinates of all the scheduled pick-up points for the day's work. It also allows the user to choose the branch from which to initiate the simulation.

Subsequently, two data frames are created with real distances and durations, consistently connecting to the external OSRM software. The solving algorithm is then executed, and the user is provided with the optimal visitation order as the output.

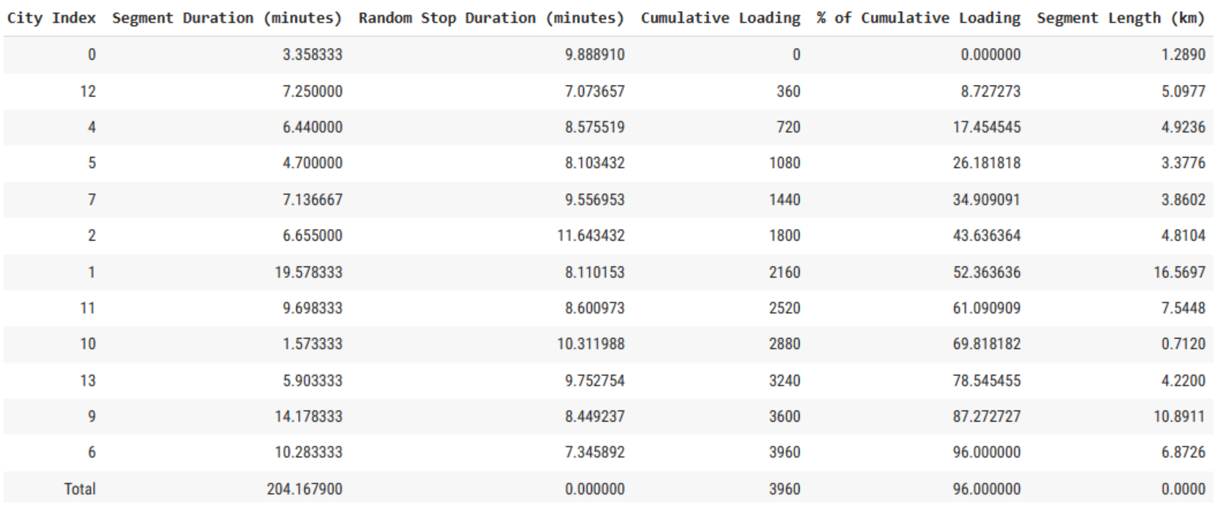
Up to this point, the program behaves exactly like the previous Last-Mile program. However, from this stage onwards, the program takes a different turn to make the simulation as close to reality as possible.

Subsequently, a check is performed at each stop to assess the chosen transport vehicle's capacity until the maximum saturation is reached. Two distinct outputs are then generated: one contains the list of points that were successfully visited, while the other comprises the stops that couldn't be made due to vehicle saturation.

In the end, a map is also created showing the final route.

5.3.2. Code Output

As can be observed from the image presented below, the output is very similar to that of the Last Mile, with one notable addition.



City Index	Segment Duration (minutes)	Random Stop Duration (minutes)	Cumulative Loading	% of Cumulative Loading	Segment Length (km)
0	3.358333	9.888910	0	0.000000	1.2890
12	7.250000	7.073657	360	8.727273	5.0977
4	6.440000	8.575519	720	17.454545	4.9236
5	4.700000	8.103432	1080	26.181818	3.3776
7	7.136667	9.556953	1440	34.909091	3.8602
2	6.655000	11.643432	1800	43.636364	4.8104
1	19.578333	8.110153	2160	52.363636	16.5697
11	9.698333	8.600973	2520	61.090909	7.5448
10	1.573333	10.311988	2880	69.818182	0.7120
13	5.903333	9.752754	3240	78.545455	4.2200
9	14.178333	8.449237	3600	87.272727	10.8911
6	10.283333	7.345892	3960	96.000000	6.8726
Total	204.167900	0.000000	3960	96.000000	0.0000

Figure 12: Pick-up Code Data Output

In this case, the 'Cumulative Duration' column is no longer displayed as it is of limited significance. Instead, two new columns have been introduced specifically for this scenario. The new columns are 'Cumulative Loading' and '% of Cumulative Loading,' which represent the progressively accumulated load on the chosen mode of transportation and the percentage of the current load in relation to the maximum load that the vehicle can sustain. These columns are utilized to determine when the courier can no longer accommodate additional cargo, marking the completion of their delivery route.

Also, the map showing the complete route will be displayed.

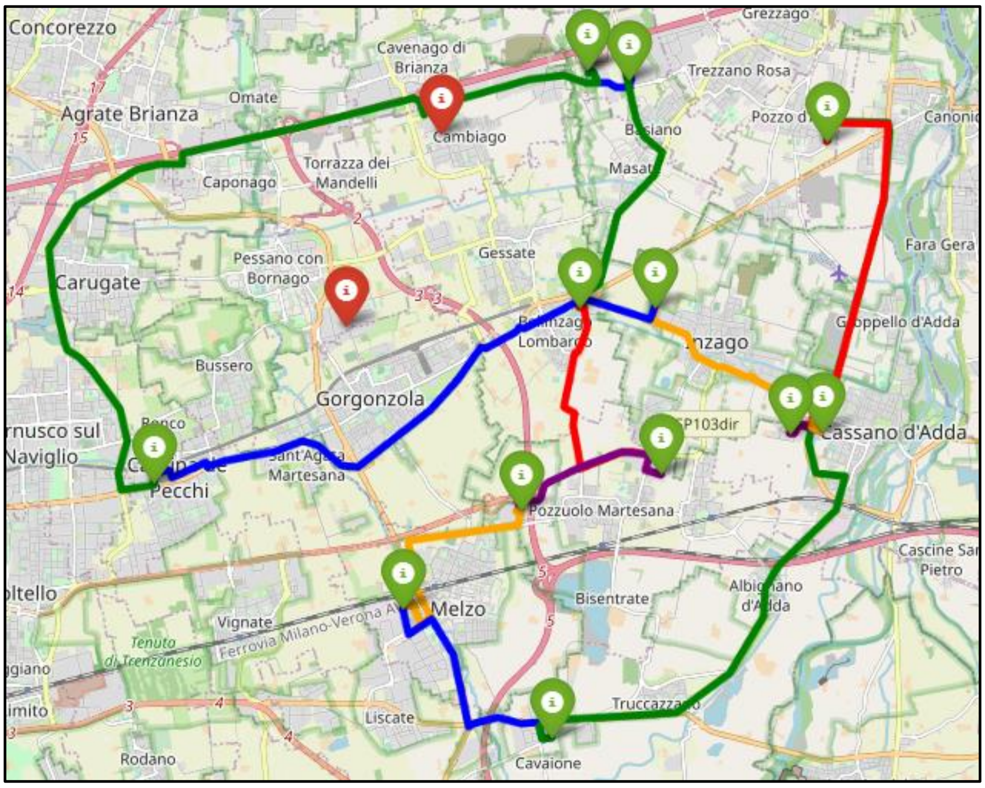


Figure 13: Pick-up Code Map Output

5.3.3. Code in detail

In this section, a detailed analysis of the program's key code lines will be conducted to grasp the code's operations without delving excessively into specific details, considering the code's similarity with the previous one.

```
# Declare variables
max_size_load = 5500 * 0.75 # Maximum size load of the truck [dm3] *
Threshold [%]
number_of_pack_per_warehouse = 20 #Average number of packs to load at
each warehouse
package_volume = 24 # Volume of each package [dm3]
```

One of the notable additions to this code, and perhaps the most crucial, is the introduction of these three lines of code where three variables of paramount importance for the program's operation are defined.

These three variables are:
`max_size_load`: this variable signifies the maximum cargo volume that any mode of transportation can carry, expressed in cubic decimeters (dm3). It is also multiplied by a percentage to establish a safety threshold;
`number_of_pack_per_warehouse`: this variable represents the average number of packages collected at each stop;
`package_volume`: this variable stands for the average volume of a retrieved package, expressed in cubic decimeters (dm3).

```
if cumulative_loading + number_of_pack_per_warehouse * package_volume
> max_size_load:
            print("Size load exceeded. Stop Loading")
            break
```

In this part of the code, the check about the truck capacity is done and when the threshold is exceeded the program stops and prints the stop where the size load is exceeded.

5.3.4. Functionalities and Limitations

The main functionalities of this program will be explained in more detail.

Starting from the beginning, the duration of stops has been increased compared to the simulation code for Last Mile delivery. For private addresses, the time had been set to range from 2 to 4 minutes, while, in this case, for warehouses, it has been extended to a range of 7 to 12 minutes. This extended duration is justified by the fact that the courier must perform multiple actions beyond a simple stop at a residential address. They need to enter a company's premises, typically collect more than one package, and navigate through larger and more complex facilities, often requiring interactions with multiple staff members, documentation, and security protocols.

Another difference lies in the number of points and their spatial distribution. In the last-mile delivery phase, the points are significantly more numerous and concentrated within a relatively limited area. However, in the pick-up process, the number of points is reduced, and they are spread across a wider geographic area.

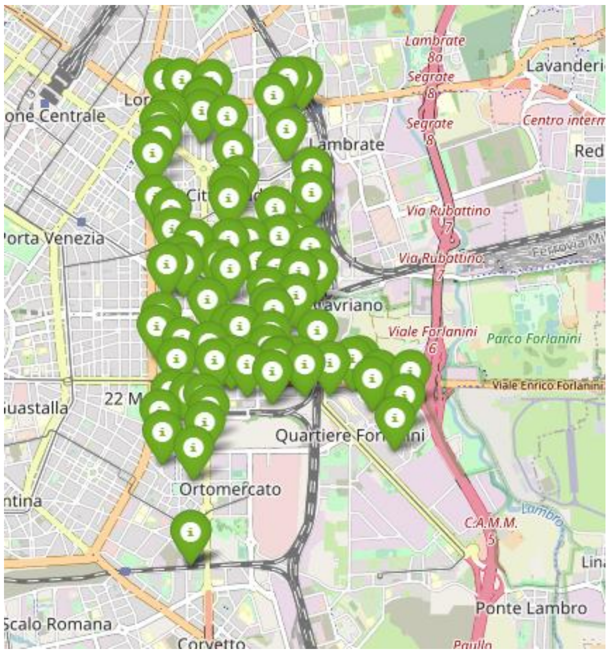


Figure 14: Delivery Points in Last-mile

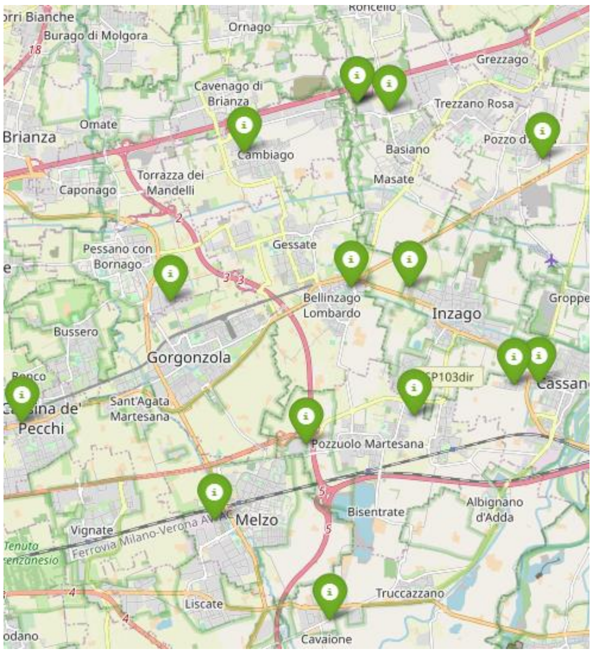


Figure 15: Delivery Points in Pick-up

This distinction in the number and spatial distribution of points serves to reflect the inherent nature of the two phases. In last-mile delivery, the focus is on efficiently reaching numerous customer addresses within a specific local region, which necessitates a higher density of points in a confined area. On the other hand, the picking phase involves a broader geographic range, often encompassing various warehouses and stores.

Regarding the setup of variables related to the maximum volumetric capacity of the mode of transportation, the number of packages collected at each stop, and the average volume of a package, it has been decided to declare them in a way that allows for easy modification based on user requirements. This approach is aimed at promoting maximum flexibility and enhancing the user experience.

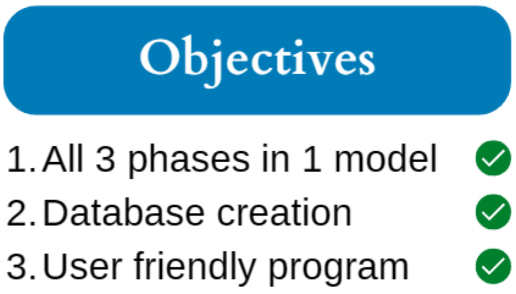
As for the '`max size load`,' a decision has been made to incorporate a threshold that is multiplied by the maximum volumetric capacity. This approach is chosen for safety and convenience reasons, as it ensures that the capacity is never fully saturated at 100%, allowing for a buffer to accommodate high-priority packages that may need to be added at the last moment.

6 Conclusion

6.1. Objectives achieved

The main objective, as previously presented in the preceding chapters, was to create a model capable of simulating and modeling all three key phases of the delivery process (first mile, last mile, and pick-up). In addition to this primary goal, there were two equally important sub-objectives: first, to establish a comprehensive database containing information on the facilities of major Italian courier companies, which would serve as the foundation for the program; and second, to ensure that the usage of this model is as user-friendly as possible, thereby enabling a broader audience to utilize it and making it highly adaptable.

As explained previously, these objectives have been fully achieved, as the created model successfully satisfies all three of the predefined goals. Moreover, the program has been designed to be as flexible as possible, allowing for continuous updates and potential improvements in the future, which will be discussed in the next section.



Objectives

1. All 3 phases in 1 model ✓
2. Database creation ✓
3. User friendly program ✓

Figure 16: Objectives Scheduled

6.2. Work limitations and Future Improvements

As explained in more detail in the previous chapters, the only limitations encountered during the project's development are related to the inability to perform simulations at different time frames and the execution time, particularly in the Last Mile program.

Regarding the first limitation, it restricts the ability to observe and subsequently analyze the impact of different days of the week and various time slots on delivery performance due to factors such as traffic and other variables.

Concerning the second limitation, as mentioned earlier, the computation time is relatively high but not excessive, considering the volume of data and requests that the software must handle. However, the total duration of 75/80 minutes (as presented in the previous Last Mile case) can pose an obstacle to daily program usage, as the waiting time becomes somewhat impactful.

Studying these limitations, however, allows for the speculation and envisioning of new additions and improvements that can be incorporated in the future to make the model as perfect and efficient as possible.

In particular, concerning the limitation of the inability to perform simulations at various time frames, a potential future enhancement could involve a shift from OSRM to Google's APIs for obtaining real distances and durations. Google's APIs allow for the adjustment of various variables, such as simulation time, but they also come with certain drawbacks, including the fact that they are not free. For this project, OSRM was chosen as the software due to these considerations.

Regarding the program's execution time, it's worth noting that the tests were conducted on a standard commercial computer. Consequently, it is likely that more advanced and powerful computers could yield better results. Additionally, the program has been designed with an emphasis on simplicity rather than performance. Therefore, there is ample room for improvement to optimize the code for maximum performance and significantly reduce the program's execution time.

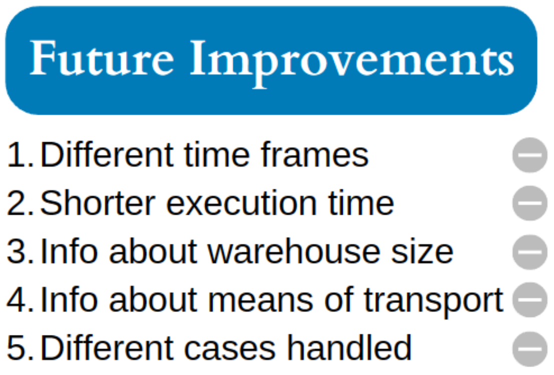
Before presenting other future improvements, it's important to consider this project as foundational groundwork for numerous more specific and intriguing projects. Therefore, the potential improvements and the areas in which this project can be applied are quite extensive.

Starting with the data collection regarding courier facilities and the subsequent creation of the database, it could be beneficial to also gather information about the dimensions of each warehouse owned by the company. This would allow for a more

accurate calculation of the environmental impact of the entire delivery process for each company.
Additionally, closely related to this, it would be valuable to know the types of transportation vehicles each company uses for each phase of the process. This information would enable precise emissions calculations and, consequently, a more detailed assessment of the environmental impact generated.

Shifting the focus to the First Mile phase, as a deeper understanding of the policies adopted by different companies is gained, it becomes relatively straightforward to incorporate various scenarios into the code, outlining how different cases are handled. For example, the current code considers scenarios in which the departure branch matches the destination branch, thus avoiding unnecessary movements through the hub. Numerous other cases can be readily integrated at different points in time to ensure that the model and the subsequent simulation closely mirror reality.

Moving on to the pick-up phase, it could be beneficial in the future to incorporate the idea that the dimensions and the number of packages collected at each pick-up point are not fixed but vary, making this aspect more faithful to reality.


Future Improvements

1. Different time frames
2. Shorter execution time
3. Info about warehouse size
4. Info about means of transport
5. Different cases handled

Figure 17: Future Improvements

In conclusion, this project has the potential to be highly valuable in assessing not only the environmental impact of the delivery processes employed by various Italian courier companies but also for a multitude of other applications. This insight can help identify key areas for improvement, possibly involving the companies themselves, to enhance the model and ultimately, to improve the impact on our surroundings and various other aspects.

Bibliography

ACI (2017) *Elenco filiali DHL Express (Italy) Srl, Automobile Club D'Italia*. Automobile Club d'Italia. https://www.aci.it/geo/v2/uploads/syc/ElencoFilialiDHL%20Express.pdf (Accessed: November 12, 2023).

Bauer, R. *et al.* (2020) *Differentiating with last-mile delivery*, *MarshMcLennan*. https://www.marshmclennan.com/content/dam/mmc-web/insights/publications/2020/june/Last-Mile-Delivery_vF_(WEB).pdf (Accessed: November 12, 2023).

Braekers, K., Ramaekers, K. and Van Nieuwenhuyse, I. (2016) 'The vehicle routing problem: State of the art classification and review,' *Computers & Industrial Engineering*, 99, pp. 300–313. https://doi.org/10.1016/j.cie.2015.12.007.

Direzione studi ricerche e statistiche dell'Autorità (2023) *Osservatorio sulle comunicazioni*. Autorita' per le Garanzie nelle Comunicazioni. https://www.agcom.it/documents/10179/30986864/Documento+generico+21-07-2023/3ac11922-32c7-4424-baf0-c51c88f08286?version=1.1 (Accessed: November 12, 2023).

Excel Maps Add-In | Geocoding API | Location Intelligence | Mapcite (2023). https://www.mapcite.com/ (Accessed: November 12, 2023).

Mangiaracina, R. *et al.* (2019) 'Innovative solutions to increase last-mile delivery efficiency in B2C e-commerce: a literature review,' *International Journal of Physical Distribution & Logistics Management*, 49(9), pp. 901–920. https://doi.org/10.1108/ijpdlm-02-2019-0048.

MWPVL International inc (2023) *Amazon global supply chain and fulfillment center network*. https://www.mwpvl.com/html/amazon_com.html (Accessed: November 12, 2023).

Sopha, B.M., Siagian, A. and Asih, A.M.S. (2016) 'Simulating Dynamic Vehicle Routing Problem using Agent-Based Modeling and Simulation,' *IEEE International Conference* [Preprint]. https://doi.org/10.1109/ieem.2016.7798095.

Staniek, M. and Sierpiński, G. (2016a) 'SMART PLATFORM FOR SUPPORT ISSUES AT THE FIRST AND LAST MILE IN THE SUPPLY CHAIN - THE CONCEPT OF THE S-MILE PROJECT,' *Scientific Journal of Silesian University of Technology. Series Transport* [Preprint]. https://doi.org/10.20858/sjsutst.2016.92.14.

Statista. (2023). *Retail e-commerce sales compound annual growth rate (CAGR) from 2023 to 2027, by country*. *Statista*. Statista Inc.. Accessed: November 12, 2023. https://www.statista.com/forecasts/220177/b2c-e-commerce-sales-cagr-forecast-for-selected-countries

eMarketer. (2022). *Retail e-commerce sales worldwide from 2014 to 2026 (in billion U.S. dollars)*. *Statista.* Statista Inc.. Accessed: November 12, 2023. https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales

Sun, L. *et al.* (2007) 'A Knowledge-Based model representation and On-Line solution method for dynamic vehicle routing problem,' in *Lecture Notes in Computer Science*. https://doi.org/10.1007/978-3-540-72590-9_32.

Tan, Y. *et al.* (2019) 'The capacitated pollution routing problem with pickup and delivery in the last mile,' *Asia Pacific Journal of Marketing and Logistics*, 31(4), pp. 1193–1215. https://doi.org/10.1108/apjml-06-2018-0217.

Van Heeswijk, W.J.A. *et al.* (2020) 'Evaluating urban logistics schemes using agent-based simulation,' *Transportation Science*, 54(3), pp. 651–675. https://doi.org/10.1287/trsc.2019.0971.

Viu-Roig, M. and Alvarez-Palau, E.J. (2020) 'The Impact of E-Commerce-Related Last-Mile Logistics on Cities: A Systematic Literature review,' *Sustainability*, 12(16), p. 6492. https://doi.org/10.3390/su12166492.

Zachariadis, E.E. and Kiranoudis, C.T. (2012) 'An effective local search approach for the Vehicle Routing Problem with Backhauls,' *Expert Systems With Applications*, 39(3), pp. 3174–3184. https://doi.org/10.1016/j.eswa.2011.09.004.

List of Figures

List of Graphs

Annex

Link for the database: Link

Link for the maps: Link

Link for the code: Link

Link for the files for the code: Link

Link for the dataset for tests: Link

Link for all: Link

Acknowledgements

Vorrei innanzitutto ringraziare le professoresse Arianna Seghezzi e Chiara Siragusa per avervi proposto questo lavoro, aver creduto in me e avermi seguito scrupolosamente durante tutta la durata del progetto.

In secondo luogo, vorrei ringraziare anche i professori Riccardo Mangiaracina e Angela Tumino per avermi fornito feedback importanti e confermato quanto di buono stessi facendo.

Un grazie alla mia famiglia per il costante sostegno e per avermi concesso la libertà di perseguire ciò che ritenevo più giusto.

Ringrazio anche tutti i miei amici per i momenti spensierati passati assieme e in particolare Daniele, che in questi 5 anni è stato più che un semplice compagno di corso.

Infine, vorrei ringraziare Gloria, semplicemente per tutto.

Grazie!