Politecnico di Torino

POLITECNICO DI TORINO

Master of Science in Mechatronic Engineering

MSc THESIS

The Drone Aided Routing Problem: Analysis of Optimization Algorithms for Last Mile Delivery

Author:
Matteo ARICÒ

Supervisor:
Prof. Giorgio GUGLIERI

Co-supervisor:
Dr. Stefano PRIMATESTA

ACADEMIC YEAR 2020/2021

POLITECNICO DI TORINO

Abstract

Mechatronic Engineering
DAUIN - Department of Control and Computer Engineering

Master of Science in Mechatronic Engineering

The Drone Aided Routing Problem: Analysis of Optimization Algorithms for Last Mile Delivery

by Matteo ARICÒ

Unmanned Aerial Vehicles are gaining momentum in many industrial sectors. While in some areas the use of drones has already become a standard practice, in some others research activities are still needed as the development has not reached a mature stage yet. This is where this Thesis project is aimed to insert in, dealing with the logistic problem of last mile delivery.
Some companies, such as Google, UPS and Amazon, have started their own programs in order to provide drone delivery in the near future. This push from established companies has led to optimism in the sector, as employing drones for this purpose looks promising. Indeed, several works in the literature handled the truck and drone routing delivery problem. Cooperation between them to deliver parcels has already been proven to be effective in terms of delivery time, costs and emissions. Most importantly, drones are extremely effective when fast speeds of delivery are necessary, as for example for medical goods shipment, since they are not subject to road network restrictions and traffic.
This work performs an analysis of some approaches already proposed in the literature in order to build an optimization algorithm able to find an optimal solution where a set of customers needs to be served by a truck and a drone. Several assumptions on the scenario need to be made and various constraints on the optimization process needs to be followed, so that a feasible solution can be obtained. Starting from this solid basis, the project is aimed at extending the scenario where a fleet of drones is employed. In this way, the solution provided by the single truck-single drone scenario is expected to be improved, even though the problem gets more complicated as the number of available vehicles increases.
Insights on the tested algorithms will be provided, displaying data relevant for the solutions analysis. The advantages in terms of completion time of the delivery process obtained with respect to the well documented single truck solution will be highlighted, as well as advantages and disadvantages among the selected approaches.
The Thesis concludes by providing possible improvements that can be made on the algorithms and suggesting future research perspectives.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Giorgio Guglieri and co-supervisor Dr. Stefano Primatesta that have greatly supported me in the completion of this project. Their weekly assistance has helped me throughout this period working on my Thesis and as I did not taken for granted this aspect, I really appreciated that and I am grateful that I worked on this project with them.
Then, obviously, I would like to mention my mom, my dad, my sister, and my whole family as they always supported me throughout this journey, that has been sometimes stressful, sometimes hard, but I hope that I gave them some satisfactions during these years.
Finally, I would like to thank all of my friends that have been with me since before I started to study at university, as they allowed to lighten these years that as I already mentioned, they revealed sometimes stressful.
Of course, I can not forget to mention friends that I made in the classrooms during this journey, as we started to know each other by sharing the struggle of passing the exams, but we ended up sharing experiences and passions that enriched me as a person and see things with different perspectives. Furthermore, I believe that, together with my family, all my friends have contributed to my personal growth during these crucial years, and I see this aspect as the most valuable thing that path has left to me.

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**
1.1 Thesis Goal and Structure . . . 2

2 Last Mile Delivery UAV Operations - State-of-the-Art **5**
2.1 What Last Mile Delivery is . . . 5
2.2 Examples of UAVs Parcels Delivery Systems . . . 6
2.3 Multi-modal Problems Literature Review . . . 8
2.3.1 Truck and Drones Delivery Classification . . . 8
2.3.2 TSP-D Literature Review . . . 10
2.3.3 TSP-D with Multiple Drones Literature Review . . . 12
2.3.4 Barriers for UAVs Massive Employment . . . 13

3 Problem Formulation **15**
3.1 Theory behind the Traveling Salesman Problem . . . 15
3.2 Terminology . . . 16
3.3 Notation . . . 17
3.4 Problem Assumptions and Formulation . . . 18
3.4.1 Assumptions . . . 18
3.4.2 Mathematical Formulation . . . 19

4 The Algorithm **23**
4.1 Local Search Algorithm - TSP-D Solution . . . 23
4.1.1 Local Search Algorithm Main Body . . . 23
4.1.2 Savings function . . . 24
4.1.3 Cost Truck function . . . 25
4.1.4 Cost UAV function . . . 27
4.1.5 Perform Update function . . . 28
Time Update function . . . 29
4.2 Practical Example & Possible Improvements . . . 30
4.2.1 Practical Example in a Simple Real Case Scenario . . . 30
4.2.2 Possible Improvements & Alternative Approaches . . . 32
4.3 Hybrid Genetic Algorithm - TSP-D solution . . . 33
4.3.1 The *Only Feasible* Solutions Variant . . . 34
Parents Selection function . . . 34
Child Generation function . . . 35
Restore function . . . 36
Select Survivors function . . . 37
4.3.2 The *Infeasible* Solutions Variant . . . 37
Split function . . . 39

Repair function 41
Select Survivors function 41
4.3.3 The *Only Feasible Variant* versus *Infeasible Variant* Comparison . 42
4.3.4 The Real Case Scenario Example solved with the *HGA - Infeasible Variant* 43
4.4 Multiple Drones Implementation 44
4.4.1 Local Search Algorithm - Multiple UAVs Adaptation 44
4.4.2 Hybrid Genetic Algorithm - Multiple UAVs Adaptation 50
The *Only Feasible* Solutions Variant 50
The *Infeasible* Solutions Variant 54
4.4.3 The Real Case Scenario Example solved with *HGA Infeasible Variant - Multiple Drones Adaptation* 55
4.5 mFSTSP Exact Method - An Alternative Approach for Multiple Drones 56

5 Results **61**
5.1 Test 1 - Algorithms Comparison - Ten Customers Scenario 63
5.2 Test 2 - Monte Carlo Simulation for Heuristic Algorithms 67
5.3 Test 3 - Algorithms Comparison - Twenty Customers Scenario 69
5.4 Test 4 - Evolution test for Genetic Algorithms 72

6 Conclusions **75**

Bibliography **77**

List of Figures

1.1 The same problem leads to different solutions depending on the vehicles employed. The goal of the Thesis is to investigate an efficient solution in order to integrate the truck delivery process assisted with drones. . . . 2
1.2 The Gantt chart shows the operational times characterizing the delivery process. The truck delivering parcels without assistance is depicted by *TSP*, the truck travel travel when assisted by one drone is depicted by *Truck* and the drone that assists the truck is denoted as *Drone*. . . . 3

2.1 The *Prime Air* drone design launched in 2019 at the conference re:MARS, photo by Jordan Stead. . . . 7
2.2 Classification of drones utilization in routing problems. Taken from [Macrina et al., 2020]. [10] . . . 9

4.1 Example of the algorithm operating in a scenario where seven customers need to be served. Some of them are eligible for drone delivery while others can be served only by the truck. . . . 23
4.2 An example on a scenario where the `cost truck` function can operate 26
4.3 Example of the solution produced by the algorithm in a real case scenario where seven customers need to be served. In this instance, every node has been considered as eligible for drone delivery. . . . 31
4.4 The Gantt chart showing the operational times in the TSP solution versus TSP-D solution. . . . 31
4.5 A `child generation` function practical example. . . . 36
4.6 A `split child` function practical example. . . . 40
4.7 The differences of performance for the *Only Feasible Variant* versus the *Infeasible Variant*. It is clear that the latter is able to find better results. . 43
4.8 Example of the solution produced by the algorithm in a real case scenario where seven customers need to be served. In this instance every node has been considered as eligible for drone delivery. . . . 43
4.9 The Gantt chart showing the operational times in the TSP solution versus TSP-D solution. . . . 44
4.10 An example on how the `cost truck` function works for the case where $v = 2$. . . . 47
4.11 Example of how algorithm works. . . . 49
4.12 Example of the solution produced by the algorithm in a real case scenario where seven customers need to be served. In this instance every node has been considered as eligible for drone delivery. . . . 55
4.13 The Gantt chart showing the operational times in the mFSTSP solution, with $v = 2$. . . . 56

5.1 The TSP solution for the instance considered, where ten customers must be served. 63
5.2 TSP-D solution of *Local Search Algorithm*. 63
5.3 TSP-D solution of *HGA - Only Feasible Variant* algorithm. 64
5.4 TSP-D solution of *HGA - Infeasible Variant* algorithm. 64
5.5 mFSTSP solution of *Local Search Algorithm - Multiple Drones*. 65
5.6 mFSTSP solution of *HGA Only Feasible Variant - Multiple Drones*. 65
5.7 mFSTSP solution of *HGA Infeasible Variant - Multiple Drones*. 66
5.8 mFSTSP solution of *Exact Method - Multiple Drones*. 66
5.9 Results of the Monte Carlo simulation, expressed in % savings of time with respect to the TSP solution produced by the solver. 68
5.10 The TSP-D solution provided by the *HGA - Infeasible Variant* for the instance created for Test 3. 70
5.11 The Gantt chart describing timing between the UAV and the truck achieved with the solution presented in Figure 5.10. 70
5.12 Results of the Monte Carlo simulation, expressed in % savings of time with respect to the TSP solution produced by the solver. 71
5.13 Evolution tendency of *HGA* algorithms over $\gamma = 10000$ iterations. . . . 72

List of Tables

2.1 Truck and Drone characteristics, suggesting promising tandem application in the delivery process, from [Chung et al.,2020]. [5] 6
2.2 Summary of reviewed literature. 13

4.1 Input data for the algorithm. 30
4.2 Data of the *Only Feasible Variant* versus *Infeasible Variant*. The numbers express the makespan of the delivery process in *seconds*. These are results obtained for γ = 10000 generations and n = 20 elements in the initialized population. 42

5.1 Test parameters. 61
5.2 Input data for the algorithm. 62
5.3 The results in terms of the optimized variable, the *completion time* of the delivery process, of the various algorithms that has been tested. . . 67
5.4 Statistical data for the Monte Carlo simulation made for the algorithms discussed. These data have been obtained over $n = 100$ iterated tests. 68
5.5 Set of coordinates generated randomly via `http://www.geomidpoint.com/random/` in a rectangular map comprehending the Rome urban area. 69
5.6 The results in terms of the optimized variable, the *completion time* of the delivery process, of the various algorithms that has been tested. . . 71
5.7 Statistical data for the Monte Carlo simulation made for the algorithms discussed. 72
5.8 Results obtained setting $\gamma = 100000$ as stop condition for the genetic algorithm. As it is clear from this Table, leaving more time for the evolution of genetic algorithms leads to improved solutions. 73

List of Algorithms

1 Local Search Algorithm . . . 24
2 Calculate savings function . . . 25
3 Calculate cost truck function . . . 26
4 Calculate cost UAV function . . . 27
5 Perform update function . . . 28
6 Time update function . . . 29
7 HGA Only Feasible Solutions Variant algorithm . . . 34
8 Parents selection function . . . 34
9 Child generation function . . . 35
10 Restore function . . . 36
11 Select survivors function . . . 37
12 HGA Infeasible Solutions Variant algorithm . . . 38
13 Split function . . . 39
14 Repair function . . . 41
15 Select survivors function . . . 42
16 Local search function for multiple drones . . . 45
17 Calculate cost truck function for the mFSTSP problem . . . 46
18 Perform update function for the mFSTSP problem . . . 48
19 HGA Only Feasible solution Variant for the mFSTSP problem . . . 50
20 Split function for the mFSTSP problem . . . 51
21 Create UAV sortie function for the mFSTSP problem . . . 52
22 Insert infeasible customers function for the mFSTSP problem . . . 53
23 Truck sub-routes update function for the mFSTSP problem . . . 53
24 Split function of Infeasible Solutions Variant for the mFSTSP problem . . . 54
25 Exact method algorithm, inspired by [Murray and Raj, 2019] [14] for the mFSTSP problem . . . 57
26 Calculate cost truck function for the mFSTSP problem . . . 58

List of Abbreviations

AI	**A**rtificial **I**ntelligence
API	**A**pplication **P**rogramming **I**nterface
CVP-D	**C**arrier **V**ehicle **P**roblem with **D**rones
DDP	**D**rone **D**elivery **P**roblem
FAA	**F**ederal **A**viation **A**dministration
FSTSP	**F**lying **S**idekick **T**raveling **S**alesman **P**roblem
GPS	**G**lobal **P**ositioning **S**ystem
GUI	**G**raphic **U**ser **I**nterface
HGA	**H**ybrid **G**enetic **A**lgorithm
HGVNS	**H**ybrid **G**eneral **V**ariable **N**eighborhood **S**earch
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
IQR	**I**nter**Q**uartile **R**ange
LTL	**L**ower **T**ruck **L**imit
mFSTSP	**m**ultiple **F**lying **S**idekick **T**raveling **S**alesman **P**roblem
MILP	**M**ixed **I**nteger **L**inear **P**rogramming
NP	**N**ondeterministic **P**olynomial time
PDSTSP	**P**arallel **D**rone **S**cheduling **T**raveling **S**alesman **P**roblem
R&D	**R**esearch and **D**evelopment
REST	**RE**presentation **S**tate **T**ransfer
RP-D	**R**outing **P**roblem with **D**rones
TSP	**T**raveling **S**alesman **P**roblem
TSP-D	**T**raveling **S**alesman **P**roblem with **D**rone
UAV	**U**nmanned **A**erial **V**ehicle
VMT	**V**ehicle **M**iles **T**ravelled
VRP	**V**ehicle **R**outing **P**roblem
VRP-D	**V**ehicle **R**outing **P**roblem with **D**rones

Chapter 1

Introduction

The technology development has recently allowed the spread of Unmanned Aerial Vehicles (UAVs), commonly referred as drones, among completely different sectors. It is possible to classify three main areas where UAVs can operate:

- **Civilian.** One of the most promising application in this area is constituted by transportation. However, this presents several safety problems related to ground risk, especially if UAVs fly over other transportation infrastructures.
  The commercial use for the building sector has become an interesting application as well, since it is possible to use UAVs for aerial photography, surveying, monitoring and inspecting the construction site. This allows to avoid dangerous scenarios and speed up the data collection with the main drawback of dealing with local regulation restrictions.
  An interesting application that is gaining popularity recently is the employment of UAVs in agriculture. In this sector, they are mainly used to spray chemicals or water over crops, or just to monitor them.
  Disaster management is another area where UAVs have already been efficiently applied. This spans from image collection to transport of essential goods for humanitarian purposes. The main drawbacks are related to limited payload and battery life.
  Surveillance is another sector where UAVs have been extensively used, either indoor or outdoor.
  Logistics is a promising area where UAVs could be applied in the near future.
  Finally, UAVs are widely used for entertainment and they are becoming a new standard to film aerial shots which are particularly suggestive for cinema production, as well as for filming events.

- **Environment.** The use of UAVs for environmental monitoring and protection has become quite common throughout the years. In particular, they have been used to monitor destructive actions that are affecting the Amazon forest or to monitor national parks. Other examples of applications are made by air quality monitoring, soil and crop monitoring and hydrology, where UAVs proven to be effective in collecting data in difficult to access areas due to adverse meteorological events.

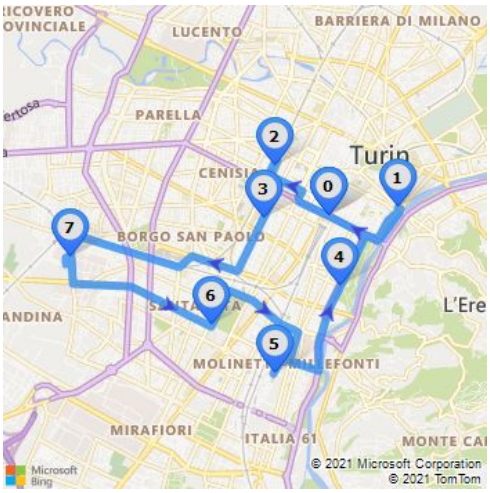
- **Defense.** The use of UAVs in this area dates back to 1982, when the U.S. navy used these devices for military applications. In this case they can either be armed to help military operations or they can protect soldiers and civilian by monitoring activities. [10]

While some applications are already in a mature stage, where the drones are used extensively and their market is steadily growing like in agricultural applications or in photography and videomaking, other fields are still placed a step behind in the

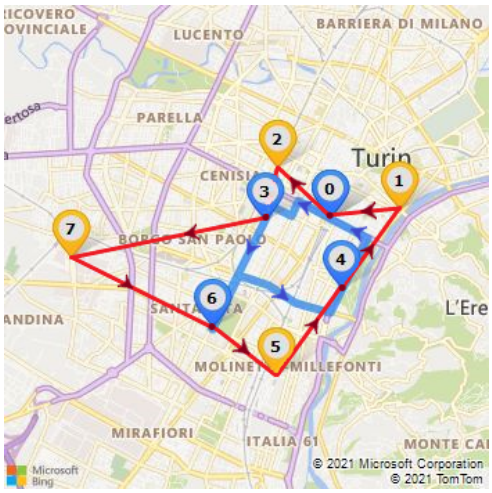
developing stage, as for example in logistics. [19]
This Thesis is aimed to investigate deeper this aspect, exploiting the already available literature, possibly adding further insights on this complex problem.

1.1 Thesis Goal and Structure

The Thesis is inserted in a broader project, where a multi-modal transportation system for urban delivery in smart cities shall be designed.
This work is aimed at investigating in depth the operational approach for the logistic problem, exploring optimization algorithms that allow to find the best solution to serve customers located at predefined coordinates through a hybrid delivery system, constituted by the truck and one or more drones. Firstly, the available literature on multi-modal delivery systems has been studied and analyzed, in order to establish the current state-of-the-art in the research field and how this problem shall be tackled. The practical aspects that shall be considered for the implementation of such system in a real case scenario has been surveyed as well, where few examples of tests and programs launched in recent years by delivery companies have been presented. Once these pieces of information have been collected, the final goal will be to implement and analyze a set of algorithms to solve the problem of a single truck and a single drone delivery. These algorithms have been selected among the most promising ones in the literature and they will be then evaluated according to various scenarios with different customer's locations instances. Eventually, they will be extended to applications where availability of multiple drones is forecast. Performing several tests, the main features of the algorithms will be highlighted, so that the best approach according to trade-offs on performance and algorithm efficiency can be selected.
The problem addressed in this Thesis is shown better in Figure 1.1.



(A) The problem consists in visiting the set of locations present in the Figure in the least amount of time. The solution reported here is referred to a single truck as delivery vehicle.



(B) The same problem of serving the customer's locations is tackled differently in this Figure. In particular the truck is helped in the delivery process by a drone.

FIGURE 1.1: The same problem leads to different solutions depending on the vehicles employed. The goal of the Thesis is to investigate an efficient solution in order to integrate the truck delivery process assisted with drones.

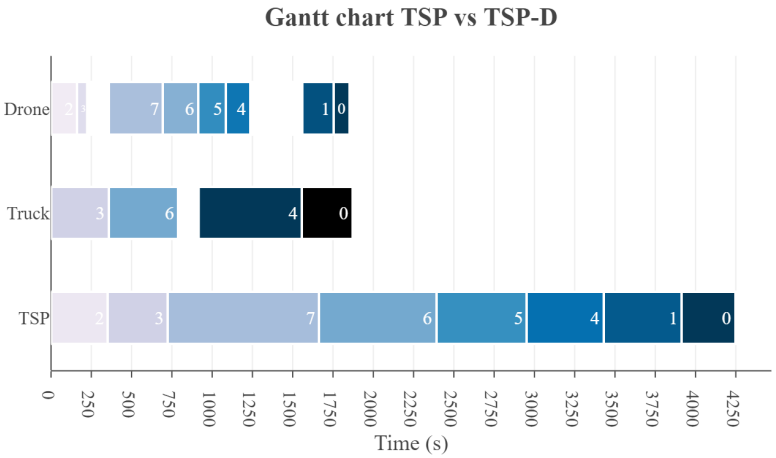


FIGURE 1.2: The Gantt chart shows the operational times characterizing the delivery process. The truck delivering parcels without assistance is depicted by *TSP*, the truck travel travel when assisted by one drone is depicted by *Truck* and the drone that assists the truck is denoted as *Drone*.

Namely, Figure 1.1a shows the conventional *Traveling Salesman Problem*, referred in the following as TSP, where a set of locations must be visited exactly once by the salesman that shall depart from and come back to the same location. The goal is to minimize a non-negative cost function associated to the entire tour, that characterizes the travel between two locations and it is generally the travel time.
Then, a drone insertion is considered so that some customers may be served by it, allowing the truck to shorten its route. The customer's assignment to the truck and to the drone must be taken carefully, because this depends on criteria based on optimization of completion time of the entire delivery tour, but it also depends on synchronization between the two vehicles and feasibility in terms of battery life and payload that can be carried by the drone. These particular aspects are depicted in Figure 1.2 and they will be thoroughly analyzed. These challenges are emphasized when multiple drones are added, therefore, analysis for the case where a fleet of drones is present will be carried out as well.
In **CHAPTER 2** a technical introduction to the subject will be provided. In particular, the Last Mile Delivery concept, real drone delivery tests and applications and the literature review on the TSP with one drone, called TSP-D, and multiple drones will be investigated.
CHAPTER 3 deals with the mathematical problem of the TSP. This problem belongs to a well-documented class of problems that are particularly hard to be solved with computer algorithms in a finite time. A mixed integer linear programming formulation, taken from [Murray and Chu, 2015] [13], that constitute the milestone for all the subsequent literature related to TSP-D problems, will be provided as well.
CHAPTER 4 presents the algorithms implementation in Python. Different approaches will be examined and the chosen methodologies will be described in depth. To retrieve the cost functions that will constitute the decision variable of the problem, the Bing Maps API has been used.
In **CHAPTER 5** real scenarios with different number of customers located at various distances will be implemented to test the algorithms under multiple situations. Insights on the data obtained will be given to establish how the algorithms performs in different operational conditions.

CHAPTER 6 closes the Thesis, summing up the work developed and providing possible improvements and further research opportunities.

Chapter 2

Last Mile Delivery UAV Operations - State-of-the-Art

2.1 What Last Mile Delivery is

The concept of Last Mile Delivery is referred to the very last step of the logistic process, where the parcel goes from the transportation hub to the final customer's location.
The main phases of the Last Mile Delivery process can be described as follows:

1. orders are placed by the customer and these enter in the digital system where they can be monitored;
2. orders arrive at the transportation hub and they wait to be delivered and this is where the logistic process actually begins;
3. orders are strategically designated to delivery personnel based on delivery addresses trying to achieve an optimized process;
4. orders are scanned before being loaded to delivery vehicles in order to monitor the parcel on both customer and company sides;
5. orders successfully reach the final destination and a proof of correct shipment is requested in order to update the delivery status.

In recent years, this process has witnessed an enormous increase in quality demand in a continuously growing market, especially due to the rapid spread of online shopping platforms. Usually, e-commerce retailers are required to deliver a huge number of parcels within small time windows, to several customers and possibly without charging any delivery fee. These requirements are obviously conflicting, and they shall be tackled with efficient management in order to survive in this highly competitive sector.
Furthermore, the main paradox is that even though customers are requiring fast and possibly free delivery services, the Last Mile Delivery process happens to be the most expensive one throughout the whole logistic chain. As stated in the OnFleet blog, "Last mile shipping can account for 53% of a shipment's total costs. Companies typically eat about 25% of that cost themselves, but this number is increasing as supply chain inefficiencies are becoming more and more costly." [15]
This has led companies to push in R&D projects in order to implement new technologies in the Last Mile Delivery operations to gain advantages with respect to the competition.
Among many other trends that will be affecting Last Mile Delivery in the future, there is the employment of autonomous robots and UAVs. In particular, these smart

devices would allow companies speed up the Last Mile Delivery process, cut the labor cost and potentially employ them 24 hours per day.
In the following Section, several examples of UAVs parcels delivery that have been tested throughout the years will be presented.

2.2 Examples of UAVs Parcels Delivery Systems

Drones are able to bring a significant strategic advantage in terms of delivery; indeed, they are generally faster than conventional ground delivery vehicles and they are not subject to any route restriction, unless flight-restricted areas are present. Their flexibility represents a key factor to manage the delivery time optimally, as this variable has become a critical factor. However, the employment of UAVs does not come without any disadvantage: the payload capacity and the flight range determined by the battery life are the main technological limitations to massively employ these devices. Furthermore, ground-related risks are still difficult to be managed.
To partially overcome these issues, the tandem truck-drone delivery system has been analyzed to be employed in the Last Mile Delivery process. This approach looks very promising as the drone and the truck characteristics are complementary, as summed up in Table 2.1. [5]

Mode	Speed	Weight	Capacity	Range	Energy Consumption
Drone	High	Light	One	Short	Low
Truck	Low	Heavy	Many	Long	High

TABLE 2.1: Truck and Drone characteristics, suggesting promising tandem application in the delivery process, from [Chung et al.,2020]. [5]

Indeed, the truck carrying the drone is able to effectively increase the flight range of the latter, that would have been otherwise limited to a circular area around the distribution center with a radius proportional to the drone endurance time. This last scenario would be critical, since depots should be re-located to highly populated areas, involving high costs for the companies. In the truck-drone tandem case, instead, the UAV can be brought in proximity to customers by the truck and it can perform deliveries while the truck is heading towards another customer, leading to an efficient time management of the delivery process. Moreover, the truck will be able to delivery heavy parcels, while the drone will be entitled to serve customers requiring low weight goods. [13]
To testify the increase of interest in drones utilization in logistics, in the last decade, several companies has carried out numerous experiments in order to practically use UAVs for parcels delivery.
Amazon has launched its drone delivery program *Prime Air* as early as 2013. The initial plan was to deliver parcels via UAVs departing from warehouses and traveling towards customer's location via GPS. Although that announcement has been welcomed with scepticism [13] at that time, Amazon kept testing and developing its system in dedicated platforms either in U.S. and abroad. [22] Its latest announcement on the program occurred in 2019, where they launched the drone design that will be used in *Prime Air* program. This drone "can fly up to 15 miles and deliver packages under five pounds to customers in less than 30 minutes", as mentioned in

the Amazon website. It features an hybrid design between an helicopter, leading to greater take-off and landing performances, and an airplane, leading to better stability and safety during transit conditions. It also features sophisticated AI technologies in order to increase safety that would not be guaranteed only by communication systems for situational awareness. This granted Amazon a certificate by the FAA to operate its aircraft in authorized flight areas. [11] Furthermore, the company claims that by using drones for deliveries, it will reach 50% of shipments with environmental zero-impact by 2030. [24]

FIGURE 2.1: The *Prime Air* drone design launched in 2019 at the conference re:MARS, photo by Jordan Stead.

Another interesting application of drone delivery involves a U.S startup, called Zipline, that partnered with the Rwanda government starting from December 2016 to deliver vital medical parcels in the country. In particular, "the company has dispatched more than 4000 units of blood products to 12 hospitals—red blood cells, platelets, and plasma that would have otherwise needed to travel by a treacherously tangled road network, losing precious hours in the race to save lives", as stated in the Time article. In this case, it is evident that drone technology has been leveraged for its efficiency and its fastness compared to the ground vehicles, whose limitations has been emphasized in this scenario by a poor road network. Moreover, being these devices critical for healthcare, they are designed differently from conventional quad-copters, in order to fly further and with adverse weather conditions. These characteristics are essential for reliable and always ready deliveries. Finally, these drones feature parachute landing in order to avoid that lots of people need to be trained to interact with these devices. The success of the company in Rwanda has led to expand the project in other African states and Latin America. [2] The company has recently used its technology to deliver COVID-19 vaccinations in Ghana, in order to reach rural area that would have otherwise been difficult to access. [1]
By the time of writing Zipline has actually expanded to U.S healthcare services, defense and disaster response sector, and even in retail and e-commerce, where a program partnered with Walmart has been launched in U.S. in September 2020. Walmart has partnered with end-to-end commercial drone delivery company as well, called Flytrex, in order to achieve safe, convenient and fast goods deliveries, leading to a decreased environmental impact as well.
Flytrex collaborated with the Iceland's greatest delivery company, AHA,with drones

utilized in a hybrid system in tandem with trucks. This system has proven its efficiency, reducing the delivery time operations. [5]
The first company that has obtained the FAA approval for commercial operations with drones was *Wing*, a branch of Alphabet, which was granted to fly in 2012. Its drone is able to deliver packages dropped via tether. The company is partnered with FedEx and Walgreens and it currently operates in U.S., Finland and Australia. The demand for its offered service has seen a drastic increase lately, due to the pandemic that oftentimes forced consumers to stay at home.
Other companies offering logistic services, like UPS and DHL have investigated the possibility of employing drones for e-commerce applications and for humanitarian purposes. DHL operated in 2018 in East Africa to deliver medical goods after a session of tests carried out in Germany [22], while UPS, partnered with Matternet, operates in U.S. and Switzerland to provide healthcare products. Matternet is collaborating with Mercedes-Benz as well, in order to develop a completely autonomous delivery system for e-commerce applications.
In general, since the demand for same-day delivery or even instant delivery is becoming larger and larger, the drone delivery market is expected to grow in the next decade. This growth is foreseeable also because of the large investments made by the companies cited above, that committed a large amount of resources in research activities and tests. [25] The current pandemic has contributed to boost the demand for contactless and safe distribution, which is guaranteed by this mode of delivery. Furthermore, drones have helped heavily struck places by delivering medical parcels and other essential goods for the population under lockdown. Drone regulations and available technologies constitutes an important role in the growth of this sector, as they are still behind the demand of services but they are rapidly advancing.[18]

2.3 Multi-modal Problems Literature Review

Since the employment of UAVs in logistics has gained interest from a commercial and academic standpoint, researches have been made in order to establish the feasibility of utilizing such devices in a real case scenario, analyze the optimization algorithms for truck and drone scheduling operations, investigate the costs related to such operations and find out the environmental impact related to energy efficiency management throughout the process.
As this project is directed towards analyzing a possible solution to the truck and drone delivery problem, the literature review will be focused on the mathematical formulation of the problem and optimization algorithm implementation for truck and drone coordination to place an effective parcels delivery.
First a classification of the myriad of problems that can be born from the classical TSP problem just by adding UAVs will be discussed. Then, the review will be focused on the specific problem type that will be tackled later in this Thesis.

2.3.1 Truck and Drones Delivery Classification

Routing problems with UAVs are classified according to how many drones are available and how they are employed for the delivery process. The classification is shown in Figure 2.2.

The routing problem with drones, RP-D, is divided in four categories:

- ***TSP-D**: TSP with drones*, that combines truck and one or multiple drones. In particular, this problem has been introduced systematically by [Murray and

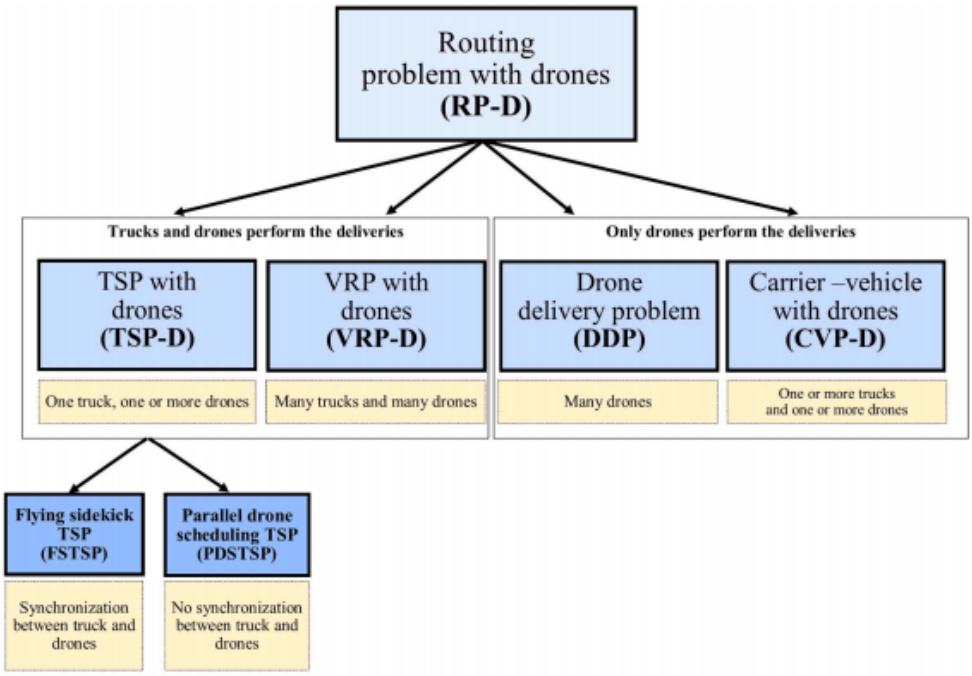


FIGURE 2.2: Classification of drones utilization in routing problems. Taken from [Macrina et al., 2020]. [10]

Chu, 2015] [13]. They provided an exhaustive mixed integer linear programming formulation over which several other approaches are built. They proposed two variants to tackle this problem:

- ***FSTSP**: flying sidekick TSP*, where a single truck and a single drone are used to carry out the delivery of the parcels. The goal is to minimize the completion time of the overall process, by visiting all the customers only once with either of the two vehicles and coming back to the depot. The drone is subjected to several constraints, such as battery life that limits the travel time and distance, the limited payload capacity and other constraints related to the coordination of activities with the truck.
  Among all the RP-D problems, it is the most documented in literature and it presents many sub-variants, classified according to the specific initial hypothesis and constraints imposed. The algorithms proposed to tackle this problem are numerous as well.
- ***PDSTSP**: parallel drone scheduling TSP*, where the goal is again the minimization of the completion time of the delivery process in which all the customers must be served by either a drone or the truck. However, in this case, there can be one drone or a fleet of drones that operate asynchronously with respect to the truck operations. Therefore, they perform one or multiple travels starting and ending to the depot and customers needs to be within the flight range from the depot. The truck will instead operate independently, reaching all the customers that can not be served by UAVs.
  Also in this case, several models and algorithms are described in literature in order to solve this type of problem.

- ***VRP-D**: VRP with drones*, it constitutes a generalization of the *TSP-D* problem, where one or multiple trucks and drones are available. It has been introduced more recently due its more complicated nature. For this reason, the problem has been firstly considered with some relaxed constraints on the drones operations. Then, restrictions have been progressively added in order to obtain a realistic framework and variants have been documented as well in literature, where several specific problems, related not only on minimization of the time but of the costs, energy consumption and so on, have been addressed.

- ***DDP**: drone delivery problems*, that can be considered as a variant of the *VRP-D*, where trucks are eliminated and the fleet is composed by multiple drones. Being this type of problem related to only drones, the approaches presented in literature are more focused towards drones characteristics, such as energy consumption, limited payload and flight range and battery charge, that may hinder the successful completion of a delivery mission. This is, however, the least documented problem in literature since only drone deliveries without any assistance by other vehicles still presents technological challenges due to the characteristics of the currently available drones.

- ***CVP-D**: carrier-vehicle problem with drones*, this problem addresses the technological difficulties mentioned in the *DDP* problem. It forecasts the employment of a team of cooperating vehicles composed by a large and slow carrier (like ships or large ground vehicles) able to transport a fleet of small and fast vehicles, that are, in this case, the UAVs. The carrier works as a sort of moving depot, carrying the parcels as well, where the fast delivery vehicles can depart and come back multiple times to serve efficiently all the customers. Constraints are usually considered for the drones, while they are in general different with respect to the *TSP-D* case for the carrier, where in this latter instance it was entitled of deliveries as well. This problem is well documented in literature and it presents several variants where different hypothesis and variables to optimize are considered.

The first two problems (*TSP-D* and *VRP-D*) belong to the bigger class of routing problems where either the truck or the drones can perform the deliveries of the parcels, while the last two (*DDP* and *CVP-D*) belong to the class where only drones can perform deliveries, while the truck is used as a support vehicle for the drones. [10]

2.3.2 TSP-D Literature Review

A remarkable amount of literature is present for the TSP-D and VRP-D problems.
The first methodical formulation has been elaborated by [Murray and Chu, 2015] [13], where the challenge of addressing an optimal assignment of customers to a UAV delivery system working in tandem with a truck has been tackled, called the FSTSP. The scenario studied foresees a single truck and a single drone. The objective is to minimize the completion time of the delivery process. In particular, a systematic mixed integer linear programming, referred as MILP, formulation has been proposed. To handle the amount of customers that are usually present in a last mile delivery scenario, an heuristic algorithm has been developed. Indeed, exact methods are inefficient in terms of run-time, due to the NP-hard nature of the problem. The heuristic algorithm features a route and re-assign approach, where first the conventional TSP problem is solved and later the drone eligible nodes are progressively

assigned to the UAV, removing them from the truck route. Finally, the savings associated to the UAV assignments and their feasibility are evaluated. A set of drone eligible nodes is established by criteria such as the impossibility of delivering heavy packages. Actual road distance metric is used to calculate the truck cost function while the Euclidean distance is used for drone travels. The assumptions performed are related to the nodes, that can be visited only once, either by a truck or a drone, synchronization between the truck and the drone that must occur, the drone that can be launched and retrieved only in customer's nodes and constraints on flight endurance that must be respected as well. Another problem formulation has been considered in this paper, that deals with the PDSTSP, where multiple drones operate in a circular area around the depot delimited by their flight range, while the truck delivers parcels independently according to a TSP route, thus no synchronization is needed. Anyhow, the objective still consists in minimizing the delivery time.
In [Ha et al., 2018] [9], the minimization of completion time of the delivery process is accounted for in parallel with the minimization of the cost of transportation, called the min-cost TSP-D. The mathematical formulation and problem assumptions are based on the [Murray and Chu, 2015] [13] formulation, with the main difference that waiting times between the truck and drone are accounted for, since they degrade the performances in terms of cost minimization. The approach is based on an hybrid genetic algorithm with adaptive diversity control. Also in this case, an initial TSP solution is considered, called the *giant tour chromosome*. The TSP tour is built by selecting two parents and cut them in two random points of the chromosome in order to mix the candidates solutions and possibly improving them by creating a fitter child for the next generation. Then, a split procedure is performed so that a drone deliveries chromosome and truck deliveries chromosome are obtained. The individual evaluation is carried out by computing the *penalized cost* $\Phi(P_1)$, i.e. the sum of the travel times between two locations for the entire tour weighted by constraints violation, and diversity of population $\Delta(P_1)$, so that a balance between diversification and intensification of solutions will be obtained in the generations, i.e. the convergence to a solution will not be fast, but the search of that solution will not be performed randomly.
[Ponza, 2016] [16] exploited the [Murray and Chu, 2015] [13] formulation, slightly modifying the assumptions on the ready-time of the truck and drone, accounting for additional time to carry out maintenance services, to build an innovative framework based on simulated annealing approach. This method is based on an initial solution to which an energy value is associated with, according to its configuration, and then its energy is decreased by a so called *cooling schedule* implemented in the algorithm, so that the solution converges towards progressively lower energy states. Other algorithms, like ant colony optimization and a naïve approach have been explored by the author, nevertheless the simulated annealing approach has been implemented and effectiveness for the considered application has been extensively proven.
The FSTSP formulation has been utilized by [Freitas and Penna, 2020] [6] as well, but they proposed a solution obtained via variable neighborhood search. The initial step consists in producing the TSP solution through an already available solver. Then, an initial solution of the TSP-D approach is generated by an algorithm based on the [Murray and Chu, 2015] [13] heuristic, where truck customer's nodes are turned into drone customer's nodes. This first tentative TSP-D solution is finally given to the general variable neighborhood search that improves the final outcome of the algorithm. In the paper, seven neighborhood structures are discussed.
The [Bouman and Agatz, 2018] [3] proposes a completely different approach based

on a newly designed mathematical model. This introduced some important assumptions, like that the drone is α times faster than the truck and the truck can wait in the same node while the drone is performing its delivery task, i.e. the drone can be launched and recovered in the same node. Unlike the FSTSP approach, this new mathematical formulation can not be handled by conventional solvers, and the initial TSP solution is obtained by exploiting the Bellman-Held-Karp dynamic programming algorithm. The TSP-D solution is obtained starting from the initial TSP solution either via another dynamic programming approach, or by A^* implementation. These methodologies, being different from the heuristic approaches seen above, produces an exact solution. In this paper, it is also highlighted that by restricting the number of visits that the truck can perform while the drone is airborne, the time required to produce the solution is significantly reduced, without affecting much the overall quality of the solution.
Other revised articles deal with the environmental impact of the drone delivery over the truck delivery, like the [Goodchild and Toy, 2017] [8]. The [Chiang et al., 2019] [4] proposes a genetic algorithm that solves the TSP-D problem by exploiting a specifically designed integer programming formulation that accounts for carbon emissions of the hybrid system and the total delivery costs.

2.3.3 TSP-D with Multiple Drones Literature Review

While still a great amount of material is present in the literature for the problem involving multiple drones, the review becomes faceted as many possible variants can be considered, depending on how many vehicles are employed, how they operates in tandem and the initial hypothesis utilized. For example, the already mentioned [Murray and Chu, 2015] [13] PDSTSP formulation already employs multiple UAVs, nonetheless the characteristics of this problem are quiet different from a scenario featuring a truck that carries multiple drones and it is utilized as a moving depot, where drones and parcels can be managed.
In [Murray and Raj, 2019] [14], the concept elaborated by [Murray and Chu, 2015] [13] is extended. Indeed, in this paper a single truck and multiple heterogeneous drones are considered. The newly proposed mathematical programming formulation, called the mFSTSP, leverages the core elements that was present in the FSTSP formulation, featuring the suitable modifications to allow the employment of a fleet of drones. The final goal is still to minimize the completion time required to deliver parcels and return to the depot. In this article, two variants on the classical time restrictions that are present for synchronization between the UAVs and the truck are present:

- *truck not required at depot*, i.e. UAVs can be launched and retrieved independently of the truck, if the depot is considered;
- *automated launch and recovery system*, i.e. the driver is not engaged in launch and delivery operations.

The approach provides a three phase heuristic algorithm where first the customers are separated in customers eligible for truck deliveries and customers eligible for drone deliveries. Then, the drone travels are assigned by defining for each customer eligible for drone delivery the launch node and the rendezvous node. Finally, time activities and scheduling of launch and retrieval operations are performed in the third phase. The work presents its results by testing the algorithm with instances with 10 and up to 100 customers to be served and five possible ways to model the

drones endurance limits are introduced.
The [Ferrandez et al., 2016] [12] proposes a different formulation of the optimization function, that is solved first by applying a K-means clustering algorithm to find optimal launch locations for drones and then a genetic algorithm is designed to solve the TSP problem connecting the launch nodes. The goal is to establish the time and energy savings of the truck and drones tandem system compared to the standalone truck or drone deliveries and to establish the optimal number of drones necessary to carry out the assigned task.
Finally, [Di Puglia Pugliese et al., 2020] [17] formalizes the problem via a programming mathematical formulation for the VRP, the DDP and the VRP-D. These three delivery configurations are then tested according to a CO_2 emission model in order to establish which delivery configuration is more sustainable in terms of environmental impact.
In Table 2.2, the reviewed paper are summarized, highlighting their main characteristics.

Paper	Formulation	# trucks	# drones	Objective function	Proposed solution
[Murray and Chu, 2015] [13]	FSTSP PDSTSP	1	1 n	Completion time	Route and re-assign heuristic
[Ha et al., 2018] [9]	FSTSP	1	1	Completion time Travel cost	Hybrid genetic algorithm
[Ponza, 2016] [16]	FSTSP	1	1	Completion time	Simulated annealing
[Freitas and Penna, 2020] [6]	FSTSP	1	1	Completion time	Hybrid general variable neighborhood search
[Agatz and Boumann, 2018] [3]	Dynamic programming	1	1	Completion time	A^*
[Goodchild and Toy, 2017] [8]	VMT	1	1	CO_2 emissions	-
[Chiang et al., 2019] [4]	Integer programming	1	1	CO_2 emissions Travel cost	Genetic algorithm
[Murray and Raj, 2019] [14]	mFSTSP	1	n	Completion time	Three steps heuristic
[Ferrandez et al, 2016] [12]	Hybrid's Netwon method with difference equation	1	n	Completion time Energy	K-means clustering with Genetic algorithm
[Di Puglia Pugliese et al., 2020] [17]	Integer programming	n	n	CO_2 emissions	-

TABLE 2.2: Summary of reviewed literature.

2.3.4 Barriers for UAVs Massive Employment

Despite the huge interest that has raised around the implementation of drone-based systems in logistics, still the presence of some barriers must be taken into account. According to the nature of the difficulties that could be faced, the following division of possible issues related to drones employment can be made:

- *privacy issues*, which becomes critical as drones are becoming more and more capable of recording and storing a huge amount of data. This problem involves particularly drones for surveillance applications;
- *security issues*, since drones are part of a telecommunication network where several sensitive data might be shared, thus being subject to cyber attacks;
- *safety issues*, directly related to accident that might be caused by adverse weather, malfunctions or hacking attacks that will cause an impact of the drone to the ground, exposing people to a possible impact with the drone;
- *technological issues*, dealing mainly with battery life, flight range and payload capacity, which are the main limitations for massive drone employment in logistics;

- *environmental issues*, related to sound pollution and air-space congestion;
- *socio-economic aspects*, where several jobs might be taken down by the utilization of autonomous vehicles, which then require more specialized workforce. [5]
- *regulation issues*, as no global guidelines on commercial utilization of drones is present and institutions are struggling to keep up to the technological advancement pace.

Chapter 3

Problem Formulation

3.1 Theory behind the Traveling Salesman Problem

Problems are characterized by their difficulty to be solved. In particular, a problem is said to be *tractable* if there exists an algorithm that is able to find the exact solution in polynomial time, i.e. given an input of size n, the run-time of the algorithm will be $O(n^k)$, for some constant k. Instead, a problem is said *intractable*, or *hard*, if the time required to be solved is superpolynomial.
According to this criteria, problems are divided in classes, and the P and NP classes will be considered. A problem belonging to the P class can be solved in polynomial time. NP class consists, instead, of problems that can be *verified* in polynomial time, i.e. if a solution of this problem is somehow available, its correctness can be checked in polynomial time of the size of the input problem. P is contained in the NP set, thus any problem in P belongs to NP as well. The main question is whether P=NP, and this is still an unsolved question in computational sciences. A particular set of NP problems is made by *NP-complete* set. This type of problems have an *unknown* status, i.e. no one has found an algorithm able to solve them in polynomial time, but a proof that no polynomial time algorithms exist to solve these problems has not been found yet. Therefore, NP-complete problems are considered at least as *hard* as any other NP problem. Nowadays NP-complete problems are considered *intractable*, as it would be surprising that, given the large number of problems belonging to this set, no one was ever able to find an algorithm solving any of these problems in polynomial time, even though no proof is given.
For these reasons, NP-completeness property establishes how hard a problem is and how unlikely is that an efficient algorithm to solve this problem exists.
NP properties actually refer to *decision problems*, i.e. all the problems whose answer is simply *yes* or *no*. However, decision problems can be directly linked to *optimization problems*, such as the traveling salesman problem, therefore NP-completeness consequences can directly affect this type of problems as well.
The traveling salesman problem can be formulated as a set of n cities, or *nodes*, that the salesman wishes to visit exactly once, finishing the *tour* in the city where he started from. Each travel between two cities is characterized by a non-negative cost, and the salesman wishes to perform the *tour* that leads to the minimum total cost possible, being the total cost the sum of individual costs due to the travel between the cities. The problem such formulated has been proven to be NP-complete.
[20] The traveling salesman problem with drones has actually been proven to belong to the *NP-hard* problem set, which is informally the set of problems at least as hard as the hardest problems in NP. [21]
Nonetheless, the intractability of those problems does not mean that they can not be solved in practice. Indeed, approximation algorithms can be applied in these cases to obtain the optimal solution in a reasonable amount of time. These algorithms will

be investigated in the next Chapter.
As a matter of fact, this Thesis has by no means the ambition to treat the problem from a mathematical standpoint, however the reasons why this problem is hard to solve needed to be clarified and thus they have been reported above.

3.2 Terminology

In the followings a list of terms and definitions that will be used throughout the Thesis will be provided:

- *TSP-D*: traveling salesman problem with drone, in particular a single truck-single drone set up is considered with this term.
- *FSTSP*: flying sidekick traveling salesman problem, it is the TSP-D formulation proposed by Murray and Chu.
- *mFSTSP*: multiple flying sidekick traveling salesman problem, where a single truck and multiple drones are available for the delivery task.
- *truck node*: it is a node exclusively served by the truck.
- *drone node*: it is a node exclusively served by the drone.
- *truck route* or *tsp tour*: it is the ordered sequence of nodes visited by the truck, that leads to the minimum cost possible for the set of customers considered.
- *sub-route*: it is a sub-set of the truck route delimited by a node where the drone is launched and a node where the drone is retrieved.
- *truck sub-route*: it is the set comprehending all the sub-routes present for a given truck route.
- *truck only nodes*: they are the nodes that can not be served by a drone because, for example, the parcel associated with these nodes exceed the payload that can be carried by the drone.
- *drone eligible nodes*: they are the nodes that can be served either by a truck or by a drone, thus, in the algorithms, they are susceptible to assignments to drone checks.
- *drone sortie*: it is a tuple $\langle i, j, k \rangle$ that describes drone operation of being launched from the truck at node i, delivering the parcel at node j and being recovered by the truck at node k. For a drone sortie, the following conditions must hold:
  - the drone can not be launched from the node i corresponding to the ending depot node;
  - the node j must be eligible for drone delivery and it must be different from the launch node i;
  - the node where the drone is retrieved k may be either the ending depot node or a customer node different from i and j and the drone travel time to accomplish the route $i \longrightarrow j \longrightarrow k$ must not exceed the drone endurance limit. [13]
- *drone travel*: it describes any drone operation moving from one node to another one. [5]

3.3 Notation

The following notation will be employed for the mathematical formulation of the truck and drone routing problem:

- $C = 1, 2, ..., c$ represents the set of all customers that must be served either by a truck or a drone;
- $C' \subseteq C$ denotes the sub-set of customers that may be served by a UAV;
- $N = 0, 1, ..., c + 1$ denotes the set of nodes that must be visited exactly once in the delivery process. Although the drone and truck must return to the initial depot node 0, in this set the final depot is depicted by $c + 1$, therefore nodes 0 and $c + 1$ actually correspond to the same location;
- $N_0 = 0, 1, ..., c$ stands for all the nodes from which vehicles can depart from;
- $N_+ = 1, 2, ..., c + 1$ represents all the nodes to which vehicles can head to during the tour;
- τ_{ij} denotes the time necessary for the truck to travel from node $i \in N_0$ to $j \in N_+$;
- τ'_{ij} denotes the correspondent time necessary for the drone to travel from node $i \in N_0$ to $j \in N_+$;
- s_L represents the service time required to *launch* the drone form a customer's node;
- s_R represents the analogous service time required for drone *recovery* at a customer's node;
- ϵ depicts the drone flight endurance, i.e. the drone battery life, which consequently determines its flight range;
- δ_t is the time necessary to physically deliver the parcel to a customer;
- M is a sufficiently large number;
- $\langle i, j, k \rangle$ depicts a drone sortie, with the characteristics defined in Section 3.2;
- P is the set of all possible drone sorties that can be feasibly flown by the UAV;
- $x_{ij} \in \{0, 1\}$ is the decision variable representing whether the truck travels from node $i \in N_0$ to $j \in N_+$, in this case setting it equal to one, or not, setting instead the decision variable to zero;
- $y_{ijk} \in \{0, 1\}$ analogous decision variable for the drone, that is set to one if the UAV flies from $i \in N_0$ to $j \in N_+$ without the truck, and it is recovered at node $k \in \{N_+ : \langle i, j, k \rangle \in P\}$;
- t_j is the time at which the truck arrives at node $j \in N_+$;
- t'_j is the time at which the drone arrives at node $j \in N_+$;
- $t_0 = t'_0 = 0$ depicts the earliest time at which the drone and the truck may leave the initial depot;

- $p_{ij} \in \{0, 1\}$ represents another decision variable that establishes whether customer $i \in C$ has been visited some time before $j \in \{C : j \neq i\}$ in the truck route. As a consequence $p_{0j} = 1, \forall j \in C$, since the depot must be the starting node of the tour;
- $1 \leq u_i \leq c + 2$ denotes an integer variable specifying the position of the $i \in N_+$ in the truck's tour. [13]

3.4 Problem Assumptions and Formulation

The single truck-single drone routing delivery problem, denoted in the following by the TSP-D problem, can be informally described by a set of customers that need to be visited exactly once either by the truck or by the drone, that operate in coordination. Some customers can not be feasibly served by the UAV; this may be due to customers assigned to heavy parcels that exceeds the payload capacity of the drone, customers located in places where landing becomes challenging or requirement of customer's signature. These nodes must be necessarily served by the truck, thus they will be included in the *truck only nodes* set. The truck and the drone must depart and return to the depot exactly once. The depot has been considered as a single location, thus, as the conventional TSP problem, the two vehicles must end their tours where it started. The UAV can travel towards customers transported by the truck, in which case its battery life is conserved, or it can perform a travel accomplishing a drone sortie, draining the energy available from the battery. The drone sortie consists in three locations from which the drone is launched, it delivers the parcel, and it performs the rendezvous maneuver. To be feasible, this travel must not exceed the drone endurance limit. In the meantime, the truck may go directly from the launch customer to the rendezvous customer, or it can perform multiple deliveries, always accounting for synchronization constraints, which imposes anyhow the truck to be at the rendezvous node before the battery life of the drone expires. During the whole delivery cycle, the drone may perform multiple drone sorties. A drone sortie may be either started from the depot or from a customer's node, where the UAV is loaded of the parcel it needs to deliver. The rendezvous node may be either another customer's node or the final depot, where in this latter case, no synchronization is needed assuming that personnel is always available to recover the drone, independently from the truck. In the launch and recovery nodes, the driver of the truck is also entitled of delivering the parcel to the corresponding customer. Therefore, several service times are taken into account. In particular, a service time required to load the parcel and possibly changing the battery of the drone is required. Then, a service time accounting for physical delivery of the parcel by the truck's driver is considered. Finally, a recovery service time is required in order to allow the driver to recover the landing drone at rendezvous node.
The goal will be to minimize the completion time of the delivery cycle, called the makespan, determined by the arrival of both vehicles at the final depot. [13]

3.4.1 Assumptions

The following assumptions have been formulated in order to coherently tackle the problem:

- the UAV can only be launched from the depot or customer's nodes served by the truck. This is a reasonable assumption as performing rendezvous at

intermediate locations would require landing on a moving truck, that is still technologically challenging, or parking on an arbitrary spot on the road side, that may not be always possible; [3]

- while the drone may visit only one customer per sortie, in the meantime the truck may visit several customers;
- the UAV is assumed to remain constantly airborne during the drone sortie, even when waiting for the truck at the rendezvous node. Therefore, if the truck arrives later than the drone at node $k \in N_+$, the drone will hover in place until it will not be recovered. In this way, the drone does not have the chance to save battery in its delivery tasks;
- if the drone is recovered at customer's node i, it may be re-launched from the same node i, however, it shall not return back to it, since the truck can not visit a customer's location twice to recover the UAV;
- non-customer's nodes visits are not allowed, as well as either truck or drone customers re-visit;
- once the UAV reaches the depot at the end of a drone sortie, it can not be re-launched from it. [13]
- the battery is assumed to be recharged anytime the drone comes back to the truck after having performed a delivery task, even if a residual battery life is still present.

3.4.2 Mathematical Formulation

The problem is formally defined by the mathematical formulation, taken from the [Murray and Chu, 2015] [13], provided below:

$$Min\ t_{c+1} \tag{3.1}$$

$$s.t.$$

$$\sum_{\substack{i \in N_0 \\ i \neq j}} x_{ij} + \sum_{\substack{i \in N_0 \\ i \neq j}} \sum_{\substack{k \in N_+ \\ (i,j,k) \in P}} y_{ijk} = 1 \quad \forall j \in C \tag{3.2}$$

$$\sum_{j \in N_+} x_{0j} = 1 \tag{3.3}$$

$$\sum_{i \in N_0} x_{i,c+1} = 1 \tag{3.4}$$

$$u_i - u_j + 1 \leq (c+2)(1 - x_{ij}) \quad \forall i \in C,\ j \in \{N_+ : j \neq i\} \tag{3.5}$$

$$\sum_{\substack{i \in N_0 \\ i \neq j}} x_{ij} = \sum_{\substack{k \in N_+ \\ k \neq j}} x_{jk} \quad \forall j \in C \tag{3.6}$$

$$\sum_{\substack{j \in C \\ j \neq i}} \sum_{\substack{j \in C \\ (i,j,k) \in P}} y_{ijk} \leq 1 \quad \forall i \in N_0 \tag{3.7}$$

$$\sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C \\ (i,j,k) \in P}} y_{ijk} \leq 1 \quad \forall k \in N_+ \tag{3.8}$$

$$2\,y_{ijk} \leq \sum_{\substack{h\in N_0\\ h\neq i}} x_{hi} \;+\; \sum_{\substack{l\in C\\ l\neq k}} x_{lk} \quad \forall i \in C,\; j \in \{C:\; j\neq i\},\; k \in \{N_+:\; \langle i,j,k\rangle \in P\} \tag{3.9}$$

$$y_{0jk} \leq \sum_{\substack{h\in N_0\\ h\neq k}} x_{hk} \quad \forall j \in C,\; k \in \{N_+ : \langle 0,j,k\rangle \in P\} \tag{3.10}$$

$$u_k - u_i \geq 1 - (c+2)\left(1 - \sum_{\substack{j\in C\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall i \in C,\; k \in \{N_+ : k \neq i\} \tag{3.11}$$

$$t'_i \geq t_i - M\left(1 - \sum_{\substack{j\in C\\ j\neq i}} \sum_{\substack{k\in N_+\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall i \in C \tag{3.12}$$

$$t'_i \leq t_i + M\left(1 - \sum_{\substack{j\in C\\ j\neq i}} \sum_{\substack{k\in N_+\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall i \in C \tag{3.13}$$

$$t'_k \geq t_k - M\left(1 - \sum_{\substack{i\in N_0\\ i\neq k}} \sum_{\substack{j\in C\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall k \in N_+ \tag{3.14}$$

$$t'_k \leq t_k + M\left(1 - \sum_{\substack{i\in N_0\\ i\neq k}} \sum_{\substack{j\in C\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall k \in N_+ \tag{3.15}$$

$$t_k \geq t_h + \tau_{hk} + s_L\left(1 - \sum_{\substack{l\in C\\ l\neq k}} \sum_{\substack{m\in N_+\\ \langle k,l,m\rangle \in P}} y_{klm}\right) + s_R\left(\sum_{\substack{i\in N_0\\ i\neq k}} \sum_{\substack{j\in C\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) + \delta_t +$$
$$-M(1 - x_{hk}) \quad \forall h \in N_0,\; k \in \{N_+ : k \neq h\} \tag{3.16}$$

$$t'_j \geq t'_i + \tau'_{ij} + \delta_t - M\left(1 - \sum_{\substack{k\in N_+\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall j \in C',\; i \in \{N_0 : i \neq j\} \tag{3.17}$$

$$t'_k \geq t'_j + \tau'_{jk} + s_R - M\left(1 - \sum_{\substack{i\in N_0\\ \langle i,j,k\rangle \in P}} y_{ijk}\right) \quad \forall j \in C',\; k \in \{N_+ : k \neq j\} \tag{3.18}$$

$$t'_k - (t'_j - \tau'_{ij}) \leq e + M(1 - y_{ijk})$$
$$\forall k \in N_+,\; j \in \{C : j \neq k\},\; i \in \{N_0 : \langle i,j,k\rangle \in P\} \tag{3.19}$$

$$u_i - u_j \geq 1 - (c+2)p_{ij} \quad \forall i \in C,\; j \in \{C : j \neq i\} \tag{3.20}$$

$$u_i - u_j \leq -1 + (c+2)(1 - p_{ij}) \quad \forall i \in C,\; j \in \{C : j \neq i\} \tag{3.21}$$

$$p_{ij} + p_{ji} = 1 \quad \forall i \in C,\; j \in \{C : j \neq i\} \tag{3.22}$$

$$t'_l \geq t'_k - M(3 - \sum_{\substack{j\in C\\ \langle i,j,k\rangle \in P\\ j\neq l}} y_{ijk} - \sum_{\substack{m\in C\\ m\neq i\\ m\neq k\\ m\neq l}} \sum_{\substack{n\in N_+\\ \langle l,m,n\rangle \in P\\ n\neq i\\ n\neq k}} y_{lmn} - p_{il}$$

$$\forall i \in N_0,\ k \in \{N_+ : k \neq i\},\ l \in \{C : l \neq i, l \neq k\} \tag{3.23}$$
$$t_0 = 0 \tag{3.24}$$
$$t'_0 = 0 \tag{3.25}$$
$$p_{0j} = 1 \quad \forall j \in C \tag{3.26}$$
$$x_{ij} \in \{0,1\} \quad \forall i \in N_0,\ j \in \{N_+ : j \in i\} \tag{3.27}$$
$$y_{ijk} \in \{0,1\} \quad \forall i \in N_0,\ j \in \{C : j \neq i\},\ k \in \{N_+ : \langle i,j,k \rangle \in P\} \tag{3.28}$$
$$1 \leq u_i \leq c+2 \quad \forall i \in N_+ \tag{3.29}$$
$$t_i \geq 0 \quad \forall i \in N \tag{3.30}$$
$$t'_i \geq 0 \quad \forall i \in N \tag{3.31}$$
$$p_{ij} \in \{0,1\} \quad \forall i \in N_0,\ j \in \{C : j \neq i\} \tag{3.32}$$

Equation 3.1 represents the objective function that seeks to minimize the makespan of the delivery cycle. In particular, t_{c+1} is the latest time of arrival by either the truck or the UAV, i.e. it is equivalent to $min(max(t_{c+1}, t'_{c+1}))$, as constraints 3.14 and 3.15 imposes a connection between the truck time and the drone time, required for rendezvous nodes. Nonetheless, this does not mean that the truck and drone must be synchronized to come back to depot. Constraints 3.2 imposes the single visit condition for each customer, either by a truck or a drone. Constraints 3.3 and 3.4 requires that the truck starts its tour from the depot and it comes back to the depot at the end of its delivery cycle exactly once respectively. Constraint 3.5 is entitled of eliminating the truck sub-tours, with bounds on the auxiliary variable u_i provided in Constraint 3.29. The truck visiting node j, must depart from j once it performed its delivery task, and this is specified by Constraint 3.6. Constraints 3.7 and 3.8 establish that the UAV may be launched and retrieved at any node, either customer's node or depot, at most once. If the UAV is launched from node i and recovered at node k, Constraint 3.9 imposes that both nodes i and k must be assigned to the truck tour. A similar condition for the drone starting at the depot and being retrieved at node k is formulated in Constraint 3.10, while Constraint 3.11 imposes that if the UAV is launched from customer i and collected at customer k, the truck must visit i before k. Constraints 3.12 and 3.13 are aimed at coordinating the UAV launch and the truck at node i, so that the drone can not be launched from this node if both the drone and the truck are not present at the customer's location. For similar purposes, Constraints 3.14 and 3.15 are introduced, in this case coordinating the time of truck and drone at rendezvous node k. Constraint 3.16 establishes the time update for the truck tour if the truck travels from node $h \in N_0$ to $k \in N_+$. The actual arrival time of the truck at node k must include the service times necessary for drone launch or the drone rendezvous, if either of these conditions happen at node k. Furthermore, a service time necessary for physical delivery at node k is accounted for. Note that the last term ensures that, if the truck travel from h to k does not occur, then the arrival time value shall not be increased. Constraint 3.17 states that the drone launched from node i and serving node j must incorporate the travel time necessary to go from customer i to customer j. Also in the case of UAV, a service time necessary for delivery of the parcel is included. However, the service time for launch s_L is not accounted for the drone as Constraints 3.12 and 3.13 already incorporate it. Similarly, Constraint 3.18 is aimed at incorporating the drone travel time from node j to node k and in this case the service recovery time s_R is accounted for, as the drone might arrive later

than the truck at node k. The UAV endurance limitations are depicted by Constraint 3.19. The last term imposes that this Constraint becomes active if and only if the drone travel $i \longrightarrow j \longrightarrow k$ takes place. Constraints 3.20 to 3.22 establish the correct ordered truck's path. Constraint 3.23 prevents that the UAV is launched from a node l before it is retrieved at node k, as in a single drone scenario, this situation would be infeasible. This Constraint is tied with Constraint 3.26 for proper working conditions. Finally, Constraints 3.27 to 3.32 depict the domain of the decision variables.
Even if this model has been taken from [Murray and Chu, 2015] [13], as already mentioned, a slight modification has been introduced to account for service times necessary to actually deliver the parcels, that was not taken into account in the original model.
In the next Chapter the problem such formulated will be tackled by developing a set of heuristic algorithms.

Chapter 4

The Algorithm

4.1 Local Search Algorithm - TSP-D Solution

The first approach analyzed has been heavily inspired from the [Murray and Chu, 2015] [13] work, for multiple reasons: it has been among the first approaches proposed in the literature to tackle the TSP-D problem, thus it constitutes a milestone over which several other methodologies are built, and, as a first step, it is the most intuitive technique to start the analysis.
The algorithm is based on a heuristic approach that first solves the conventional TSP, and then it tries to sequentially remove nodes from the TSP tour and assign them to a drone. Possible truck route swap that would lead to an improved TSP-D solution are investigated as well. The process goes on until the algorithm is not able to find any improvement on the solution.
An example of this algorithm in action is shown in Figure 4.1.

(A) The `tsp tour` produced by the TSP solver. Grey nodes represent the depot, green nodes represent drone eligible nodes and red nodes represent truck only nodes.

(B) In a first iteration the node *4* is removed from the `tsp tour` and it is assigned to the UAV, starting from node *2* and arriving to node *3*.

(C) The final solution is obtained by assigning node *5* to the UAV that travels from node *3* and arrives at the depot *0*.

FIGURE 4.1: Example of the algorithm operating in a scenario where seven customers need to be served. Some of them are eligible for drone delivery while others can be served only by the truck.

4.1.1 Local Search Algorithm Main Body

The main algorithm is depicted in the pseudocode of Algorithm 1.
First, it is necessary to load the `.json` that contains all the needed information to calculate the cost matrix and to establish the other variables necessary to start the program, such as the drone endurance ϵ, service times s_L and s_R, nodes that can not be served by the drone and so on.
To determine the set of drone eligible nodes, C', it is necessary to remove from the total set of customers the ones that are can not be served by the UAV, depicted by the vector `truck only nodes`. In Line 2 the conventional `tsp tour` is calculated through a solver. In particular, it takes as input the complete set of customers C and it returns the `tsp tour` made by customers visited by the truck such that the travel time

is minimized, and the vector of the arrival time `t` at each customer's node. The utilized solver is *Google OR tools*, but this can be a generic solver.
It is useful to define a variable that denotes the nodes visited by the truck correlated to the drone travels as well. This will be evident in the followings as the `tsp tour` will be split in many sub tours, called `sub-route`. These depict within the `tsp tour` the truck nodes where the drone is launched and where it is recovered. The vector including all the `sub-routes` is called the `truck sub-routes`. This concept can be better clarified by Figure 4.1c, where an example of `sub-route` is `[2, 1, 7, 3]`, while the `truck sub-routes` vector is `[[0, 2],[2, 1, 7, 3],[3],[3, 6, 0]]`.
The `truck sub-routes` vector is initialized in Line 3 and it will be progressively modified throughout the code when drone assignments will be performed.

Algorithm 1: Local Search Algorithm

```
   Initialize: response, truck only nodes, ϵ, s_R, s_L, δ_t
 1 max savings = 0;
 2 C_prime = C/truck only nodes Call the tsp solver;
 3 truck sub-routes = [tsp tour]
 4 while no improvement is produced do
 5     for j ∈ C_prime do
 6         Call savings(j, t, tsp tour) function;
 7         for all sub-route into which the tsp tour is divided do
 8             if this sub-route is already assigned to a UAV then
 9                 Call cost truck(j, t, savings, sub-route) function;
10             else
11                 Call cost uav(j, t, savings, sub-route) function;
12             end
13         end
14     end
15     if max savings > 0 then
16         Call perform update(i*, j*, k*, served by uav) function;
17         Reset max savings
18     else
19         STOP;
20     end
21 end
```

Once the initialization procedure is completed, the TSP-D optimization process begins: each drone eligible node `j` is investigated by a ***for*** *loop*. In particular, the `savings` function is called in order to establish the time reduction in case of removal of the `j` node from the `tsp tour`, as it can be seen better in Algorithm 2.

4.1.2 Savings function

Calculating the `savings` value is trivial if no UAV is assigned to the current `sub-route`: `savings` value is calculated as the sum of the truck travel time from node `i` to node `j`, the delivery time δ_t associated to node `j` and the truck travel time from node `j` to node `k` minus the new truck travel time that would be employed by removing `j` from the `tsp tour`, depicted in Line 3.
It becomes more complicated if a UAV is associated to the considered `sub-route`, since truck and drone are supposed to be synchronized. In particular, if removing `j` from the `sub-route` leads to a truck waiting time at the end of it, meaning that

the drone travel takes longer time than the new reduced `sub-route` without node `j`, then this would lead to a negative `savings` time, that translates in a worsened solution with respect to the one with `j` in the `tsp tour`. Indeed, note that the algorithm is seeking for positive values of `savings` rather than negative. This calculation that accounts for synchronization between truck and drone is performed in line 9.

Algorithm 2: Calculate savings function

Input: `j, t, tsp tour`

1 **find** `i`, customer's node right before `j` in `tsp tour`;
2 **find** `k`, customer's node right after `j` in `tsp tour`;
3 `savings` = $\tau_{ij} + \delta_t + \tau_{jk} - \tau_{ik}$;
4 **if** `j` *belongs to a* `sub-route` *already served by a UAV* **then**
5 **find** `a`, first customer's node of the `sub route`;
6 **find** `b`, last customer's node of the `sub route`;
7 **find** `j'`, customer's node currently served by UAV;
8 **find** t'[b], arrival time of truck at node `b` when `j` is removed;
9 `savings` = **min**(`savings`, `t'[b]` $- (\texttt{t[a]} + \tau'_{aj'} + \delta_t + \tau'_{j'b} + s_R)$);
10 **end**
11 **return** `savings`

As an example, it is possible to refer to Figure 4.1b: a `drone sortie, (i, j', k) = (2, 4, 3)` has been assigned, leading to the `sub-route = [2, 1, 7, 3]`. Suppose that the `savings` function is evaluating the removal of `j` = 7 from its `sub-route`. If the new `sub-route = [2, 1, 3]` has still a longer duration than the drone travel $\tau'_{aj'} + \delta_t + \tau'_{j'b} + s_R$, then the `savings` value would be positive but with a decreased magnitude with respect to the trivial case, since the truck needs anyhow to recover the drone. A more critical situation happens when the new `sub-route = [2, 1, 3]` has a shorter duration than the drone travel that would lead to a wait time of the truck at node `b` = 3 and so a negative value of `savings`, worsening the overall TSP-D solution.
After having evaluated the removal of node `j` by the `savings` function, another ***for*** *loop* is initiated, in order to investigate the insertion of `j` in any of the `sub-route` that compose the overall `tsp tour`. `j` can be inserted either as a truck node, if the considered `sub-route` is already served by a UAV, through the `cost truck` function, or it can be inserted as a drone node if the considered `sub-route` is not yet served by a UAV, through the `cost uav` function.

4.1.3 Cost Truck function

The `cost truck` function aims at evaluating the insertion of the node `j` in a `sub-route` as a truck node. In particular,it will be desirable to swap a truck node when this will lead to reduced waiting times between the UAV and the truck. It is called whenever a `sub-route` contains a node `j` $\in C'$ that can not be assigned to a UAV because this is already operating on the considered `sub-route`.

FIGURE 4.2: An example on a scenario where the `cost truck` function can operate

Algorithm 3: Calculate cost truck function

Input: `j, t, savings, sub-route`

find `a`, first node of the `sub-route`;
find `b`, first node of the `sub-route`;
for ***all*** `i` $\in$ `sub-route` **do**
  find `k`, node right after `i` in the `sub-route`;
  `cost` $= \tau_{ij} + \delta_t + \tau_{jk} - \tau_{ik}$;
  if `cost` < `savings` **then**
    if t[`b`] $- t$[`a`] + `cost` $\leq \epsilon$ **then**
      if `savings - cost > max savings` **then**
        `served by uav = False;`
        $i^* =$ `i;`
        $j^* =$ `j;`
        $k^* =$ `k;`
        `max savings = savings - cost;`
      end
    end
  end
end
return i^*, j^*, k^*, `served by uav, max savings`

For example, in Figure 4.2 suppose that the `sub-route = [2, 1]` has a shorter duration than the corresponding `uav sortie = (i, j', k) = (2, 4, 1)` travel time. This will result in a drone waiting time at node `b` = 1. In any case, the drone travel time should always be smaller than the endurance limit, ϵ. By taking suitably a node in the following `sub-route` and inserting it in the considered `sub-route`, it will be possible to reduce the time of the overall TSP-D solution. For example, by taking node `j` = 3, after having calculated its `savings` related to its removal from `sub-route = [1, 3, 6, 5, 0]`, its insertion in `sub-route = [2, 1]` can be evaluated, leading to a new `sub-route = [2, 3, 1]`. However, feasibility needs to be accounted for, i.e. the truck travel time $\tau_{23} + \delta_t + \tau_{31}$ must stay below ϵ, in order not to run out of battery before the truck arrival.
The procedure that handles this scenario is shown in Algorithm 3.
i^*, j^*, k^* denote the relevant nodes necessary for the update that will be performed later in case of a convenient truck node swap is found, once each node `j` $\in C'$ has been analyzed. The `served by uav` flag determines whether the update to the TSP-D solution involves a truck node swap or a drone node assignment. In this case, since the `cost truck` function is entitled of truck node swap assignment, it is set to `False`. The `max savings` variable store the most convenient assignment among all the `j` nodes analyzed for each iteration, either for truck or drone nodes. This, indeed,

constitutes the driving factor for whether updating the TSP-D solution or not.

4.1.4 Cost UAV function

The `cost uav` function is called whenever a node `j` $\in C'$ can be potentially assigned to a drone travel depicted by a `uav sortie = (i, j, k)`. This happens if the node `j` investigated does not belong to a `sub-route` where a UAV travel has already been assigned.
The function is depicted in Algorithm 4

Algorithm 4: Calculate cost UAV function

Input: `j, t, savings, sub-route`
for ***all*** `i` $\in$ `sub-route` **do**
  for ***all*** `k` $\in$ `sub-route` *such that* `k` *comes after* `i` **do**
    if $\tau'_{ij} + \delta_t + \tau'_{jk} \leq \epsilon$ **then**
      find $t'[\mathtt{k}]$ arrival time at node `k` when node `j` is removed from the `sub-route`;
      if $t'[\mathtt{k}] - t[\mathtt{i}] + s_R + s_L < \epsilon$ **then**
        `cost` = `max(0, max(`$t'[\mathtt{k}] - t[\mathtt{i}] + s_R + s_L$`,` $\tau'_{ij} + \delta_t + \tau'_{jk} + s_R + s_L$`)` $-(t'[\mathtt{k}] - t[\mathtt{i}]))$;
      end
      if `savings - cost > max savings` **then**
        `served by uav = True;`
        $i^* = \mathtt{i}$;
        $j^* = \mathtt{j}$;
        $k^* = \mathtt{k}$;
        `max savings = savings - cost;`
      end
    end
  end
end
return i^*, j^*, k^*, `served by uav, max savings`

It is possible to refer again to Figure 4.1 in order to explain the drone travel insertion mechanism.
In the first attempt to insert a drone delivery in the `tsp tour` shown in Figure 4.1a, all the nodes `j` $\in C'$ are examined, i.e. the nodes represented by the green squared diagram blocks. At the first iteration, node `j` = 2 is considered, and if its removal leads to a reduced completion time of the TSP tour and it can be feasibly inserted in a `drone sortie`, i.e. there exists a node `i` from which the drone can be launched and a node `k` in which the drone can be recovered without violating the drone endurance limits, then the corresponding temporary solution will be stored in $i^* = \mathtt{i}, j^* = \mathtt{j}, k^* = \mathtt{k}$, with its associated savings `max savings` and the `served by uav` flag will be set to `True`. However, in the next iteration, node `j` = 4 is considered and its evaluation leads to a greater time reduction of the `tsp tour` completion, thus the previous $i^* = \mathtt{i}, j^* = \mathtt{j}, k^* = \mathtt{k}$ and `max savings` will be overwritten. For the next nodes `j` $\in C'$ all the possible insertions are evaluated. Finally, the algorithm established that the most convenient solution is to serve customer `j` = 4 with the drone starting from `i` = 2 and landing at `k` = 3. The `tsp tour` is updated as in Figure 4.1b. With a further attempt to insert a drone delivery, it has been found that node `j` = 5 can be conveniently served by drone, producing the TSP-D solution depicted in Figure 4.1c. After this drone delivery assignment, it is important to notice that no other drone travels

are possible with subsequent attempts, since all the drone eligible nodes are either inside a `sub-route` where the UAV has already been assigned or they constitute a launch or a retrieve node.
In this case, the only possible improvement could be obtained by swapping node 7 to the other `sub-route`. If no improvements are obtained by this swap, the algorithm will stop.

4.1.5 Perform Update function

Once all drone eligible nodes have been evaluated, the `perform update` function is called whenever the algorithm is able to produce an improvement of the final solution, i.e. when `max savings` > 0. This function is entitled of updating the `tsp tour`, its associated `truck sub-routes`, the UAV assignment and the `t` vector that represents the arrival time of the truck at each `tsp tour` node.
The function is shown in details in Algorithm 5

Algorithm 5: Perform update function

Input: i^*, j^*, k^*, `served by uav`
if `served by uav` **then**
  remove j^* from the `tsp tour`;
  split the `truck sub-routes`;
  update the drone delivery nodes;
  if $i^* \in C'$ **then**
    remove i^* from C';
  end
  if $k^* \in C'$ **then**
    remove k^* from C'
  end
else
  remove j^* from the `tsp tour`;
  insert j^* in the new position in the `tsp tour`;
  update the `truck sub-routes`;
end
Call the `time update(`*truck sub-routes, drone nodes*`)` function;
return `tsp tour, t, truck sub-routes`

If the `served by uav` flag is set to `True`, it means that the j^* node needs to be served by a UAV, thus the drone assignment procedure begins: in particular, it is necessary to remove j^* from the `tsp tour` and suitably split the `truck sub-routes`. These are obtained by taking the previously assigned `truck sub-routes` and divide them into the nodes served by the truck before i^*, the nodes served by the truck between i^* and k^*, without j^* and the nodes served by the truck after k^*. The drone delivery nodes are updated as well in order to take track of all the nodes served by the UAV. Furthermore, it is necessary to remove i^* and k^* from the drone eligible nodes set. This action is needed in order to prevent the algorithm to attempt to assign these nodes to the UAV, because otherwise it would not be possible anymore to launch the drone from node i^* to node j^*, or recover it at node k^*.
If, instead, a truck swap is performed, i.e. the `served by uav` flag is set to `False`, then the node j^* must be removed from the current position in the `tsp tour` and it must be placed in the correct position, between i^* and k^*. The `truck sub-routes` affected by this swap must be updated as well.
At the end of the update procedure, the `t` vector must be re-calculated.

Conversely, if the `max savings` will not be changed by any drone assignment or truck swap, it means that no improvement can be produced and the algorithm will not call the `perform update` function and it will be stopped.
Finally, the `tsp tour`, the drone deliveries and the total completion time will be printed out.
In Section 4.2 this algorithm will be further clarified with a simple real case scenario.

Time Update function

A critical aspect that has to be highlighted is the update of the `t` vector when an assignment is performed. Indeed, particular attention must be devoted to synchronization between the truck and the drone travels, as this concept is not trivial in terms of computation. For example, when a drone travel is longer than the corresponding `sub-route`, the truck must wait for the drone before leaving to serve the next customer.
This procedure is shown in Algorithm 6

Algorithm 6: Time update function

Input: `truck sub-routes, drone nodes`
1 **initialize** `t = [0` *for nodes* $\in$ *tsp tour*`]`;
2 `index` = 0;
3 **for** ***all*** `sub-route` $\in$ `truck sub-routes` **do**
4 `time truck` = 0;
5 **find** `a` first node of the `sub-route`;
6 **find** `b` last node of the `sub-route`;
7 **for** `node`$\in$ `sub-route` **do**
8 **find** `i` current `node`;
9 **find** `j` next `node` in `sub-route`;
10 `index` = `index` + 1;
11 **if** *it is the last* `node` *of the* `tsp tour` **then**
12 `time truck` = `time truck` + τ_{ij};
13 `t[index]` = `t[index - 1]` + τ_{ij};
14 **else**
15 `time truck` = τ_{ij} + δ_t;
16 `t[index]` = `t[index - 1]` + τ_{ij} + δ_t;
17 **end**
18 **end**
19 `j'` = `drone nodes` associated to the considered `sub-route`;
20 `time drone` = $\tau'_{aj'} + \delta_t + \tau'_{j'b}$;
21 **if** `time drone` > `time truck` **then**
22 `t[index]` = `time drone - time truck`;
23 `time truck` = `time drone - time truck`;
24 **end**
25 **if** `time drone` > ϵ ***or*** `time truck` > ϵ **then**
26 **INFEASIBLE!**
27 **end**
28 **end**
29 **return** `t`

As it can be seem from Line 1, the `t` vector is initialized as a vector of as many zeroes as nodes present in the `tsp tour` currently, i.e. without all the nodes that

have been assigned to the UAV. The variable `index` in initialized as well, in order to sequentially fill the `t` vector. At Line 4 the `time truck` variable is introduced. This variable is needed in order to calculate the completion time of the single `sub-route` currently considered. This will be essential to establish the feasibility of the solution. Indeed, for any `node` composing the `sub-route`, the `time truck` and `t[index]` are updated, adding to them the truck travel time τ_{ij} and the delivery time δ_t if applicable. Then, the `time drone` associated to the considered `sub-route` is calculated as the drone travel time necessary to complete the `uav sortie = (a, j', b)`. Since synchronization between the drone and the truck has been assumed, if the drone arrives later than the truck at node `b`, then the truck has to wait the drone, and this operation is accounted for in Line 21. Finally, a check on the feasibility of the solution is performed and then the time vector `t` is returned.

4.2 Practical Example & Possible Improvements

4.2.1 Practical Example in a Simple Real Case Scenario

	Coordinates	
0	45.06259	7.678623
1	45.06442	7.696024
2	45.07169	7.665551
3	45.06242	7.662471
4	45.05035	7.681623
5	45.03527	7.665348
6	45.0436	7.649661
7	45.05573	7.614408

(A) Table of the coordinates for the practical example.

0	388	292	297	301	512	618	1116
334	0	626	631	428	639	961	1450
388	625	0	311	689	901	628	1011
374	763	292	0	435	659	373	880
314	423	606	573	0	287	612	1119
707	816	970	872	418	0	612	1227
561	980	526	473	582	501	0	754
1100	1488	973	920	1087	927	668	0

(B) Truck travel time matrix, generated trough the Bing Maps API and taken as cost function matrix.

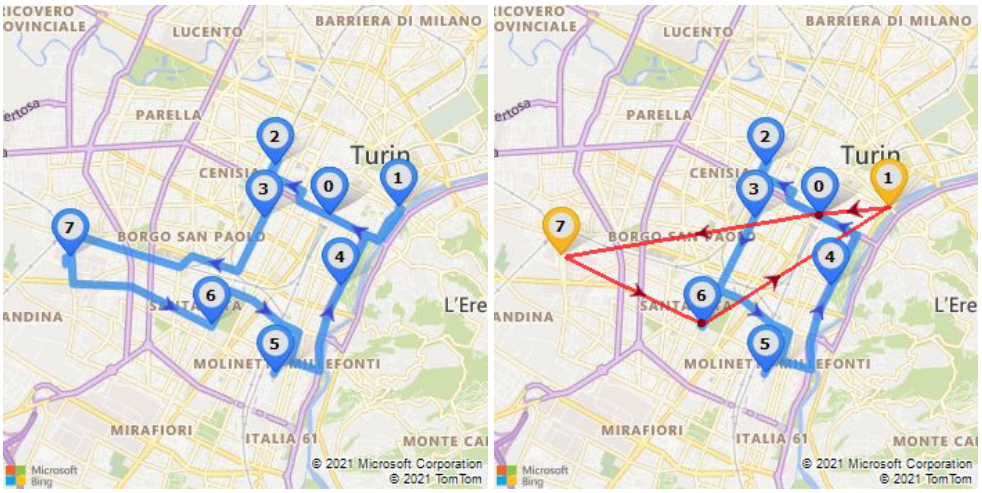
0	99	103	91	99	231	223	367
99	0	181	190	138	290	310	466
103	181	0	76	193	291	242	316
91	190	76	0	145	217	167	277
99	138	193	145	0	151	188	382
231	290	291	217	151	0	111	331
223	310	242	167	188	111	0	221
367	466	316	277	382	331	221	0

(C) Drone travel time matrix, generated trough the Euclidean distance matrix provided by the Bing Maps API and by assuming an average drone speed of 50 km/h.

TABLE 4.1: Input data for the algorithm.

Table 4.1 shows the main inputs necessary to start the algorithm. In particular, from the set of coordinates in Table 4.1a it is possible, through the Bing Maps API, to obtain the distance matrix and the travel time matrix shown in Tables 4.1b and 4.1c, that are calculated in real time based on traffic conditions. The request makes use of REST HTTP protocol. The list of coordinates are passed with a `string` format and the response is constituted by a `.json` file. This file is automatically handled

by the application in Python that extracts the relevant information for the algorithm. The request can be done either within the Python application, or through an external application. In this case, to interact with the Bing Maps API, the Postman App has been used, because of the easy to use GUI that allowed to operate easily with the API.



(A) The conventional TSP solution of the instance considered.

(B) The TSP-D solution produced by the *Local Search Algorithm*.

FIGURE 4.3: Example of the solution produced by the algorithm in a real case scenario where seven customers need to be served. In this instance, every node has been considered as eligible for drone delivery.

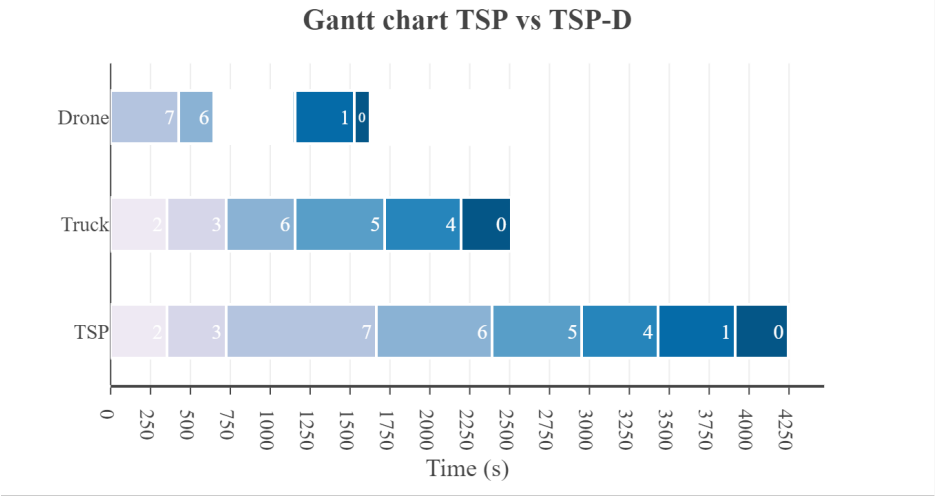


FIGURE 4.4: The Gantt chart showing the operational times in the TSP solution versus TSP-D solution.

Figure 4.3 overlays the truck route with a map showing the coordinates selected. Figure 4.3a depicts the conventional TSP tour, whose solution has been calculated by the *Google OR tools* solver, while Figure 4.3b shows the TSP-D solution calculated through the *Local Search Algorithm*. As it is evident from the map, the algorithm is able to cut efficiently from the truck route the furthest nodes, assigning them to the UAV and generating the square shaped truck route that covers much less distance

than the correspondent solution without drone. However, it is intuitive that, since in this example it has been supposed that all nodes are eligible for drone delivery, more UAV assignments could have been done in order to optimize the TSP tour. This has not been possible since, once the algorithm assign the drone to a node and to a specific `truck sub-route`, all the nodes within the `truck sub-route` can not be assigned to another UAV.
Figure 4.4 shows the truck and drone operations with respect to time. The *TSP* labelled row depicts the time employed by the truck for the conventional TSP tour, starting from the depot *0* and reaching progressively the nodes indicated on the section's right sides in which the Gantt chart is divided. The *Truck* row indicates the timing of the truck for the TSP-D solution, i.e. the solution that employs one truck and one drone, in a similar way as described for the *TSP* row. Finally, the *Drone* labelled row shows the drone operations in time, that must be coordinated with the truck in the *Truck* row. Indeed, in the first travel, the drone starts at depot *0*, it reaches node *7* and it must be retrieved at node *6*. Note, however, that in the meantime, the truck is performing other deliveries, thus the drone arrives at node *6* at time $588s$ before the truck that arrives at that node at time $1216s$. Therefore, the drone must hover at node *6* waiting for the truck for $1216s - 588s = 628s$, depicted in the chart with a blank space. This is possible as long as endurance constraints are not violated. The same happens for the next drone travel from *6*, with delivery to customer *1* and arriving at depot *0*. However, in this case, waiting constraints can be relaxed, since it is possible to assume that the drone at the depot can be directly retrieved without waiting for the truck. By a first observation it is possible to notice that, even if the algorithm would probably not converge to a global optimum, it anyhow produces a relevant reduction of the makespan of the overall solution.

4.2.2 Possible Improvements & Alternative Approaches

Based on the previous considerations made about convergence of the algorithm to a local optimum, an empirical experiment has been made during the design, in order to investigate better the algorithm. In particular, an instance made by *22* nodes plus the depot situated in Turin has been used. The *Google OR tools* solver has been set to solve the problem with the following parameters:

- `local search metaheuristic = GUIDED LOCAL SEARCH;`
- `time limit = 30 seconds`

Since this instance is relatively simple, the solver is able to find the best TSP tour consistently, so the *Local Search Algorithm*, even though it has a function that is able to recombine the truck nodes to improve the solution, it actually never used it for this particular case, because the truck route was already the best possible. Therefore, the *Local Search Algorithm* only inserted drone travels. To analyze better the behavior of the `cost truck` function entitled of swapping truck nodes, the `tsp tour` optimal solution produced by the solver has been manually modified, shuffling some nodes inside the `tsp tour`. This could be considered as an artifact to simulate the case for a more complicated scenario where the solver can not find the optimal `tsp tour` within the `time limit` set. From this experiment, the `cost truck` function has been tested successfully, producing an improvement of the overall TSP-D solution through truck nodes swap even though it never produced the same solutions that was obtained directly with the solver. However, in some cases, it turned out that the *Local Search Algorithm* that took as input a manually modified, non-optimal `tsp`

`tour` was performing better than the combination of the solver plus the *Local Search Algorithm*.
This has proven that the algorithm converges to a local minimum solution, without exploring any other possible combinations. The approach with an already built solver for the conventional TSP scenario plus the *Local Search Algorithm* allowed to simplify the problem both in the concept and in the code. However, even if the solver is able to find the optimal solution for the pure TSP problem, this does not necessarily mean that the correspondent TSP-D solution will be optimal as well. This happens because, even if the TSP problem and TSP-D problem looks very similar, they are intrinsically different, because the former is based on one vehicle only, while the latter has two vehicles available.
These considerations have given the idea to integrate the *Local Search Algorithm* into a more complex framework, as it will be discussed in the following Section.

4.3 Hybrid Genetic Algorithm - TSP-D solution

As suggested by the empirical test performed in the previously discussed algorithm, a more complex methodology shall be employed in order to find a better solution, possibly converging to a local minimum that is closer to the global one. However, the *Local Search Algorithm* will not be discarded, but it will be integrated in this new approach investigated, in order to give rise to a blended heuristic. In particular, this new algorithm has been suggested by the [Ha et al., 2018] [9] work, that proposes a genetic algorithm approach to solve the TSP-D problem. In this case, the main structure will be integrated with the *Local Search Algorithm* heuristic. For this reason, this framework is referred as *Hybrid Genetic Algorithm*, or *HGA*.
A genetic algorithm approach forecasts an evolution process, starting from an initial *population* made by *individuals*, which is able to produce progressively improved solutions based on the previous *generation*. In this case, the individuals, that are pure TSP solutions, carry the *genetic material* of previous generations, comprehending the information related to each individual that is valuable for an efficient *evolution*. The transmission of genetic material throughout the generations is made through a *crossover* process, where individuals are split and re-assembled in order to compose a new *genome* that will possibly produce an improved solution with respect to previous generations. The evolution is carried out through an iteration process, where these individuals are progressively selected according to some *fitness* criteria, that in this case will be the completion time of the delivery process. As previously mentioned, evolution mechanisms will be helped in this case by the *Local Search Algorithm*. In the followings, two variants of this algorithm will be analyzed:

- the *only feasible* solutions variant, where only feasible solutions of the TSP-D problem will be considered, i.e. solutions that do not violate drone endurance constraints;
- the *infeasible* solutions variant, where also infeasible solutions will be considered within the algorithm. However, the final solution must obviously be feasible, therefore methodologies to discourage infeasible selections will be discussed as well.

The two variants share the same framework, with the major difference that the second one can include in its population infeasible solutions, while the first is populated

only by feasible individuals. A comparison between the two methods will be provided, as well as a discussion of the advantages brought by considering infeasible individuals.

4.3.1 The *Only Feasible* Solutions Variant

The first variant of the *HGA* analyzed is shown in Algorithm 7.
As it can be seen, it features a simple structure, within which several functions are built in and they will be discussed in the followings, except for the *Local Search function* that has already been described in details in Section 4.1.

Algorithm 7: HGA Only Feasible Solutions Variant algorithm

Initialize: response, truck only nodes, $\epsilon, s_R, s_L, \delta_t$
generate random population of n individuals;
$\gamma = 0$
while $\gamma <$ `threshold` **do**
  $\gamma = \gamma + 1$;
  Call `parents selection`(*population, fit values*) function;
  Call `child generation`(*parents, drone deliveries*) function;
  Call `local search`(*child*) function;
  Call `restore`(*educated child, educated drone deliveries*) function;
  Call `select survivors`(*restored child, educated child, educated drone deliveries, population*) function;
end
Call `select fittest individual`(*population, fit values*) function;

The algorithm starts with the variable initialization necessary to establish the problem parameters. Then, a population of n individuals is initialized. This step has been performed using the *DEAP* library available for Pyhton, since it allows easier and faster management of evolutionary algorithm variables. The generation index, γ, is initialized as well and it will constitute the **STOP** condition of the algorithm. Indeed, once a predefined `threshold` on γ is reached, the algorithm will stop automatically. However, other **STOP** conditions can be implemented, such as stop after a certain number of iterations without improvement, or after a certain fitness value goal is overcome.

Parents Selection function

Algorithm 8: Parents selection function

Input: `population, fit values`
for `i` = *1:2* **do**
  sample `k` = 2 random individuals in the population;
  pick the fittest individual among the selected ones;
  assign the selected individual to `parents[i]`
end
return `parents`

The `parents selection` function select two individuals of the current population, where *individual* stands for a TSP tour made by all the customers available, without the depots attached. These selected TSP tours will constitute the basis for the child generation, that thus will inherit the genetic material from the previous generation through the parents. The selection is performed, as referred by [Ha et al., 2018] [9], through the *tournament* method, that is depicted in Algorithm 8.

Child Generation function

The `child generation` function is entitled of generating a new individual that inherits the genetic material from the selected parents. This allows the genetic algorithm to produce progressively evolved solutions, since the child will inherit the features of previous generations, possibly being different from previous individuals in order to introduce diversification and thus exploration capacity of the algorithm itself to investigate many solutions.
The algorithm is shown in Algorithm 9.
The inputs of this function are the `parents`, that are individuals selected from the population, thus they are pure TSP vectors without the depots attached, and the `drone deliveries` associated to them, that are an empty set if it is the first iteration or if a tentative TSP-D solution has not yet been calculated, while they contains the nodes served by the UAV if the correspondent individuals have previously undergone through a tentative TSP-D solution evaluation.

Algorithm 9: Child generation function

Input: `parents, drone deliveries`
insert depots to the parents;
initialize `r` $\in [0, 10]$, integer variable;
if `r` < 5 ***or*** `drone deliveries` *is empty* **then**
  consider the first parent `parents[1]`;
  sample `[a, b]` from `parents[1]` with `a` located before `b` in the TSP tour;
  assign all the nodes of `parents[1]` between `a` and `b` to the `child`;
  if `drone deliveries` *within* `a` *and* `b` **then**
    skip `drone deliveries` nodes;
  end
  assign the remaining nodes to `child` from `parents[2]`, keeping the order;
else
  sample `[a, b]` from `drone deliveries[1]` with `a` located before `b` in the `drone deliveries`;
  assign all the nodes of `drone deliveries[1]` between `a` and `b` to the `child`;
  if `parents[1]` *TSP nodes within* `a` *and* `b` **then**
    skip `parents[1]` nodes;
  end
  assign the remaining nodes to `child` from `parents[2]`, keeping the order;
end
return `child`

This function can be explained better with the help of Figure 4.5.

In this example, the `parents` = $[P_1, P_2]$ selected from the population are shown with the correspondent `tsp tour` and `drone deliveries` that are associated to them, to represent a possible final TSP-D solution. Suppose that these tentative solutions has been evaluated in a hypothetical iteration that happened before. Once the depot nodes *0* are added at the beginning and at the end of the `parents`, the child generation procedure begins.
Consider the first the case, where the randomly initialized variable is $r < 5$. In this case, the parent P_1 and its associated `tsp tour` will be taken. The nodes $[a, b]$ will be sampled randomly from the `tsp tour`, with `a` that shall be always located before

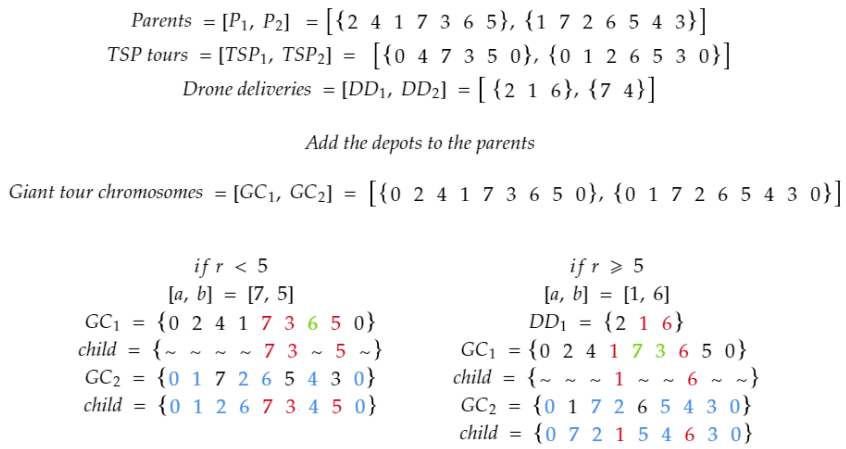


FIGURE 4.5: A `child generation` function practical example.

`b`, leading to the result `[a, b]` = $[7, 5]$. Therefore, the `giant tour chromosome`$_1$, that is basically an individual with the depots added, shall transmit the nodes between `a` and `b` to the `child`, only if they belong to the correspondent `tsp tour`. Indeed, node *6* has not been transmitted to the `child` because, even if it is located in between `a` and `b` in the `giant tour chromosome`, this belongs to the `drone deliveries` associated with P_1. It should be noted also that the positions of the transmitted nodes is kept from the `giant tour chromosome` to the `child`. To fill the empty nodes left by this procedure, the P_2 is considered with its associated `giant tour chromosome`. In particular, the nodes inherited from the vector GC_2 are kept ordered to complete sequentially the empty spaces of the `child` vector.
If, instead, the random variable $r \geq 5$ the `drone deliveries` vector, DD_1 will be considered in order to transmit the genome to future generations. Thus, `[a, b]` $= [1, 6]$ are randomly sampled from DD_1, always with `a` positioned before `b` in the `giant tour chromosome`. The latter is used in order to transmit all the nodes between `a` and `b` belonging to DD_1 keeping their positions. In this simple case, only *1* and *6* will be transmitted. Then, similarly to the previous case, the empty nodes of the `child` are filled by the ordered nodes of vector GC_2.

Restore function

Algorithm 10: Restore function

Input: `educated child, educated drone deliveries`
for `sub-route` ∈ `truck sub-routes` **do**
  find `a` first node of `sub-route`;
  find `b` last node of `sub-route`;
  insert randomly the `drone delivery` associated with the `sub-route` within `a` and `b`;
  remove the depots;
end
return `restored child`

Once the `child` has been *educated* by the *Local Search function*, the produced tentative TSP-D solution needs to be restored in order to be properly inserted in the population. This is made through the `restore` function, shown in Algorithm 10, that is aimed at producing an individual coherent with other individuals present in

the population.
Note that the vector of `truck sub-routes` should be returned by the *Local Search function*, thus it assumes the same meaning as discussed in Section 4.1.

Select Survivors function

The `select survivors` function is aimed at discarding individuals that have poor characteristics, either in terms of diversification or in terms of fitness value, represented in this case as the completion time of all the deliveries. Furthermore, it is entitled of updating the population, inserting the *educated child* solution produced in the previous steps.
This procedure is shown in Algorithm 11.

Algorithm 11: Select survivors function

Input: `restored child, educated child, educated drone deliveries, population`

if *there is one individual equal to* `restored child` **then**
  if *fitness value of* `restored child` < *fitness value of* `individual` **then**
    remove `individual` from the population;
    insert `restored child` in the population;
    update tentative solution with correspondent `educated drone deliveries`;
  else
    ignore `restored child`
  end
else
  remove `individual` with greatest fitness value;
  insert `restored child` in the population;
  update tentative solution with correspondent `educated drone deliveries`;
end
return `population`

Once the whole genetic algorithm process ends, the best TSP-D solution present in the `population` is selected through the `select fittest individual` function, whose selection is trivially based on the fitness value associated with the individual and its relative TSP-D solution.
Note that with this algorithm is not possible to produce *infeasible* solutions since the TSP-D solutions are generated directly from a pure TSP vector, that is the `child`, with the *Local Search function*, which, as defined in Section 4.1, does not consider infeasible solutions.

4.3.2 The *Infeasible* Solutions Variant

In this Section, an additional function that is able to already create a TSP-D solution within the genetic algorithm framework will be discussed. Since the `tsp tour` and `drone deliveries` related to the TSP-D solution will be generated pseudo-randomly, these will be either *feasible* or *infeasible*. In this case, the *Local Search function* will be entitled of improving the pseudo-random *feasible* TSP-D solution if possible, or to repair the *infeasible* solution according to some probability.

The general structure is similar to the *Only Feasible Variant*, however, it will be reported here for completeness. The different features of Algorithm 12 will be highlighted in the next Sections.

Algorithm 12: HGA Infeasible Solutions Variant algorithm

Initialize: response, truck only nodes, $\epsilon, s_R, s_L, \delta_t, \omega, \eta, \mu$
generate random population of n individuals;
$\gamma = 0$
while $\gamma <$ `threshold` **do**
  $\gamma = \gamma + 1$;
  Call `parents selection`(*population, fit values*) function;
  Call `child generation`(*parents, drone deliveries*) function;
  Call `split`(*child*) function;
  Call `local search`(*split child, split drone deliveries*) function;
  Call `restore`(*educated child, educated drone deliveries*) function;
  if `split child` *infeasible* **then**
    Sample random P $\in [0, 100]$ **if** $P > 50$ **then**
      Call `repair`(*restored truck route*) function;
      Call `restore`(*repaired child, repaired drone deliveries*);
      update population;
    else
      update population;
      increase ω
    end
  else
    update population;
    decrease ω
  end
  if *infeasible individuals* > η ***or*** *feasible individuals* > μ **then**
    Call `select survivors`(*population* η) function;
  end
end
Call `select fittest individual`(*population, fit values*) function;

It is immediately evident that few more variables need to be initialized: ω constitutes a penalizing factor for the travel time calculation that will be considered whenever the feasibility constraints will be violated, through the following equation:

$$t_p \;=\; t \;+\; \omega \cdot \max\left(0,\; \sum_{j \,\in\, sub-route}^{i \,=\, j-1} \tau_{ij},\; (\tau'_{ij} \;+\; \tau'_{jk})\right) \tag{4.1}$$

where t_p represents the penalized cost that will be inserted in the time vector `t` of arrival time of the truck at each node, t is the actual time employed by the truck to reach the current node considered, $\sum_{j \,\in\, s_r}^{i \,=\, j-1} \tau_{ij}$ depicts the truck travel time necessary to serve all the customers within the considered `sub-route` and $\tau'_{ij} + \tau'_{jk}$ indicates the drone travel time to complete the `uav sortie = (i,j,k)` associated with the `sub-route`. If this operation takes place, it means that either $\sum_{j \,\in\, s_r}^{i \,=\, j-1} \tau_{ij}$, or $\tau'_{ij} + \tau'_{jk}$, or both exceeded the drone endurance constraints, thus the maximum between the two will be considered as an additional term for the time calculation.

η and μ represent respectively the maximum number of *infeasible* individuals and the maximum number of *feasible* individuals allowed inside the population. For a population of $n = 20$ the following parameters have been considered for the design:

$$\omega = 1 \qquad \eta = 15 \qquad \mu = 25$$

The next steps remain unchanged, with `parents selection` and `child generation` functions operating as in Section 4.3.1. Then, the `split` function is called and, as this constitute the main novelty with respect to the *Only Feasible Variant*, it will be discussed in the next Section.

Split function

The `split` function splits the `child` vector taken as input, made by a pure TSP tour with every customer's node plus the initial and final depots, into a pseudo-random TSP-D solution where a `tsp tour` combined with `drone deliveries` are defined. In this way, the drone assignment and the relative `truck sub-routes` are not checked for feasibility, thus the produced TSP-D solution can be either *feasible* or *infeasible*. The function is shown in Algorithm 13.

Algorithm 13: Split function

Input: `child`
sample random $n_d \in [0, len(C')/2]$ = number of drone nodes;
sample n_d random `drone nodes` $\in C'$;
if *in* `drone nodes` *there are consecutive nodes as ordered in* `child` **then**
  remove previous node from `drone node`;
end
for `sub-route` $\in$ `truck sub-routes` **do**
  for `j` $\in$ `drone nodes` **do**
    if `j` *is currently in the* `sub-route` **then**
      sample random `i` $\in$ `sub-route` before `j`;
      sample random `k` $\in$ `sub-route` after `j` and before the next drone node in `drone nodes`;
      assign `j` to `drone deliveries`;
      remove `j` from `child` and C';
      if `i` $\in C'$ **then**
        remove `i` from C';
      end
      if `k` $\in C'$ **then**
        remove `k` from C';
      end
      update `truck sub-routes`;
    end
  end
end
return `split child, split drone deliveries`

To show better how this function works, refer to Figure 4.6.

The `child` is the input of the function. The vector `drone deliveries` depicts the vector of customers that will be actually served by the UAV, while `drone nodes` is referred to a tentative assignment of customers to the UAV. In particular, suppose that $n_d = 5$ is the random number of `drone nodes` that must be tentatively assigned.

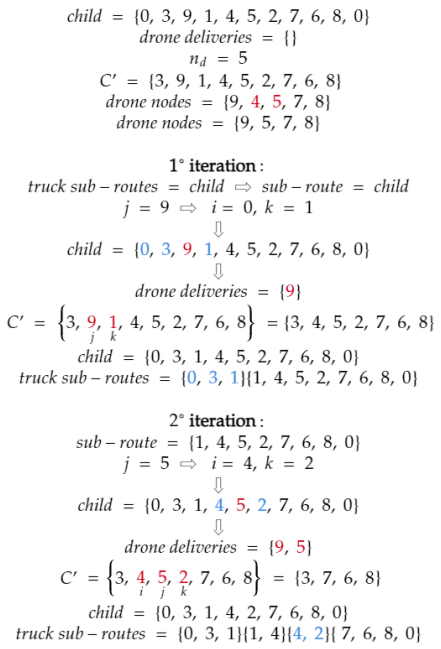


FIGURE 4.6: A `split child` function practical example.

Thus, *5* `drone nodes` are sampled from the set of drone eligible customers, C'. However, this random `drone nodes` assignment has led to two consecutive nodes as they are appearing in the `child` vector. This is not possible, since it is supposed that the UAV can make one delivery at a time and after each delivery it has to come back to the truck to be recharged and launched again. Therefore, the drone would not be able to travel to node *4* and then to node *5* without truck nodes in between that are necessary for a UAV rendezvous. For this reason, the tentative node *4* is removed from the `drone nodes` vector.
Now, the drone assignment can start by assigning consecutively `j` $\in$ `drone nodes`. In the first iteration, still no drone assignments are present, thus the `child` vector corresponds to `truck sub-routes` vector. With `j` = 9, `i` and `k` are sampled randomly, with the condition that `i` must be before `j` and belonging to the considered `sub-route` and `k` must be after `j` but before the next `drone node` *4*. Thus, the possibilities are `i` = `random`$([0,3])$ and `k` = `random`$([1,4])$. The obtained nodes are `i` = 0 and `k` = 1 in the example. At this stage, it is necessary to assign `j` to the `drone deliveries`, remove it from the `child` and update the `child` with a `truck sub-routes` subdivision. Moreover, it is necessary to remove `[i, j, k]` from C'. In this case, since `i` $= 0 \notin C'$, only `[j, k]` will be removed.
In the second iteration, `j = 5` is considered for assignment, thus, since it belongs to the `sub-route = [1, 4, 5, 2, 7, 6, 8, 0]` $\in$ `truck sub-route`, the `sub-route = [0, 3, 1]` is skipped. Again, `i` and `k` are selected with the previously mentioned criteria, giving rise to the possibilities for `i` = `random`$([1,4])$ and `k` = `random`(2). Then, similarly as described above, the `child`, `drone deliveries`, `truck sub-routes` and C' must be updated.
The process runs out once all the nodes inside `drone nodes` are successfully inserted in the TSP-D solution.
This pseudo-random TSP-D solution is then passed in the *Local Search function* in

order to assign further nodes to *drone deliveries*, if possible. This *educated* TSP-D solution will be restored with the `restore` function in order to be coherently inserted again in the `population`. However, at this stage, a *feasibility* check must be performed, in order to establish if the pseudo-random TSP-D solution *educated* is feasible. If not, it can be repaired with 50% chance by the `repair` function, that will be briefly discussed in the next Section, otherwise it will be inserted in the `population` with a penalized fitness value. If feasible, the solution will be directly inserted in the `population`.
Note that, depending on the outcome of the feasibility analysis, the ω parameter must be updated as well. In particular, if many *infeasible* solutions are produced, it means that the algorithm is falling in a wrong region of the search space, thus increasing ω will lead to more and more penalized solutions that will be immediately discarded by the `select survivors` function. Instead, decreasing ω if many *feasible* solutions are produced encourages diversification of the algorithm, that will not converge rapidly to a local minimum.

Repair function

The `repair` function is called with 50% chance, whenever an *infeasible* TSP-D solution is generated.
The function is shown in Algorithm 14.

Algorithm 14: Repair function

Input: `restored truck route`
1 **add** start and arrival depots;
2 **initialize** `drone deliveries` as an empty set;
3 Call the `local search`(*restored truck route, drone deliveries*);
4 **return** `repaired child, repaired drone deliveries`

This function simply takes the `restored truck route` vector, that is a pure TSP vector without the depots and with all customers available assigned to the truck, and this will be *re-educated* using the *Local Search function*. In this case, the tentative TSP-D solution produced by the `split child` function is ignored, while only its *restored* version will be considered. This will undergo to the *education* procedure through the *Local Search function* that, if fed by an already *feasible* TSP or TSP-D solution, will produce another improved *feasible* solution by definition.

Select Survivors function

The `select survivors` function is here discussed since it has slight modifications with respect to the version used for the *Only Feasible Variant*. Indeed, in this case, the `population` will be updated only when either η or μ constraints are violated. In particular, the former establishes the maximum number of *infeasible* individuals that can be accepted in the `population`, while the latter denotes the maximum size of the *feasible* individuals inside the `population`. Therefore, these modifications require the

function to operate as depicted in Algorithm 15.

Algorithm 15: Select survivors function

```
Input: population, η
if there are clones inside population then
    remove clones;
end
remove η − number of clones least fit individuals;
return population
```

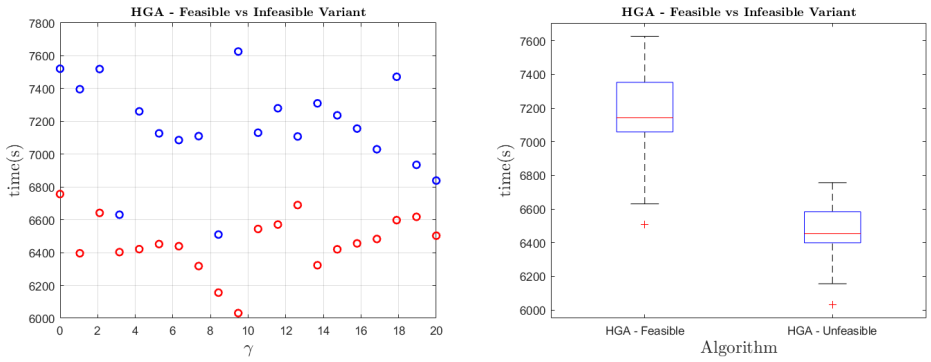
Note that a `clone` is considered as either two exact same individuals or one individual that is equal to the reverse of another individual.

4.3.3 The *Only Feasible Variant* versus *Infeasible Variant* Comparison

	Results *Only Feasible Variant*	Results *Infeasible Variant*
1	7521	6757
2	7396	6396
3	7519	6642
4	6631	6403
5	7261	6421
6	7127	6452
7	7086	6439
8	7110	6318
9	6510	6156
10	7626	6031
11	7131	6544
12	7280	6571
13	7108	6690
14	7310	6323
15	7237	6420
16	7156	6456
17	7030	6483
18	7472	6598
19	6935	6618
20	6839	6503
Average	**7164.25**	**6461.85**
Standard deviation	**279.96**	**168.28**
Minimum	**6510**	**6031**
Maximum	**7626**	**6757**

TABLE 4.2: Data of the *Only Feasible Variant* versus *Infeasible Variant*. The numbers express the makespan of the delivery process in *seconds*. These are results obtained for γ = 10000 generations and n = 20 elements in the initialized population.

In this Section, a preliminary statistical analysis in order to establish if the slightly more complicated *Infeasible Variant* is able to produce better solutions with respect to the *Only Feasible Variant* has been performed.
The results are shown in Table 4.2 and in Figure 4.7.
This improvement in performance is due to the fact that the *Infeasible Variant* inherently considers more possibilities, thus its exploration abilities in the search space in terms of possible solutions is increased. Indeed, the *Only Feasible Variant* is able to explore only a restricted set of solutions, being limited to the only feasible ones by definition.
This has been assured by [Ha et al., 2018] [9] as well, where the [Vidal et al., 2012] [23] work is cited, in which the demonstration of this behavior is analyzed in depth.



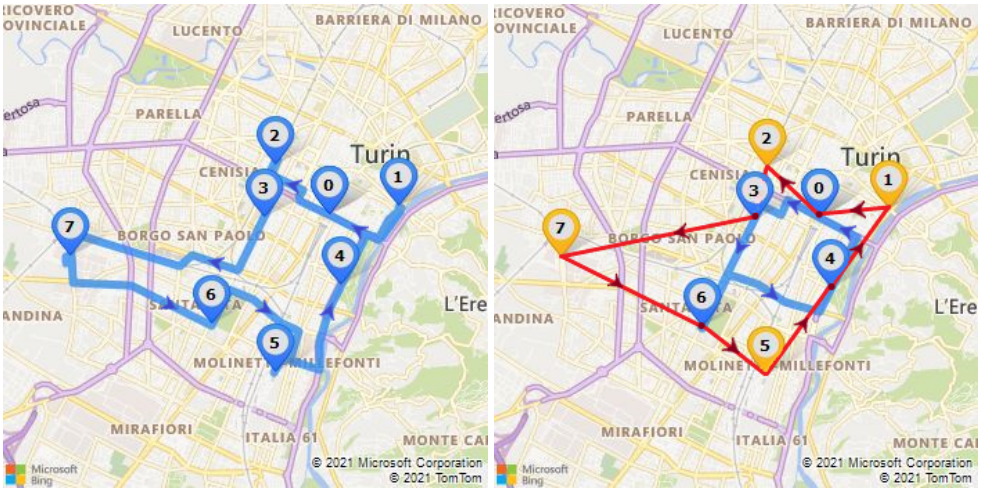
(A) The data of Table 4.2 depicted in a diagram.

(B) The data of Table 4.2 depicted with a *box and whiskers* plot.

FIGURE 4.7: The differences of performance for the *Only Feasible Variant* versus the *Infeasible Variant*. It is clear that the latter is able to find better results.

4.3.4 The Real Case Scenario Example solved with the *HGA - Infeasible Variant*

Figure 4.8 shows the solution produced by the *HGA - Infeasible Variant* of the same instance considered as example in Section 4.2, that takes as input the same coordinates and cost function matrices reported in Table 4.1.



(A) The conventional TSP solution of the instance considered.

(B) The TSP-D solution produced by the *HGA - Infeasible Variant*.

FIGURE 4.8: Example of the solution produced by the algorithm in a real case scenario where seven customers need to be served. In this instance every node has been considered as eligible for drone delivery.

It is immediately evident that the truck route completion time has been dramatically decreased with respect to the employment of the *Local Search Algorithm* only. The operational times of the truck combined with the drone can be seen from the Gantt chart in Figure 4.9.
First of all, it is important to notice that the *Local Search Algorithm* produced a solution with a makespan of 2509*s* while the *HGA - Infeasible Variant* found a feasible solution employing 1871*s*. This is testified by the tight schedule between the drone

and truck operations as well. However, to prove that with statistically meaningful data, an in depth analysis of the solutions produced with these algorithms with different scenarios will be provided in Chapter 5.
In this instance, both drone and truck leave the depot at $t = 0$. The truck goes towards customer *3* while the drone heads to customer *2* and joins the truck again at node *3*. After the service times required for the drone, this leaves again the truck going to node *7*. The truck heads to customer *6*, where it needs to recover the UAV. However, in this particular case, the truck needs to wait the drone at node *6* since the latter employs more time for its delivery. Then, the drone serves customer *5* while the truck is heading towards node *4* and finally, while the truck is coming back to the depot *0*, the drone is able to deliver the last parcel to customer *1*.
Note however that, being the *HGA - Infeasible Variant* a pure heuristic approach for the TSP-D problem, the solution produced each time the algorithm is run may slightly vary.

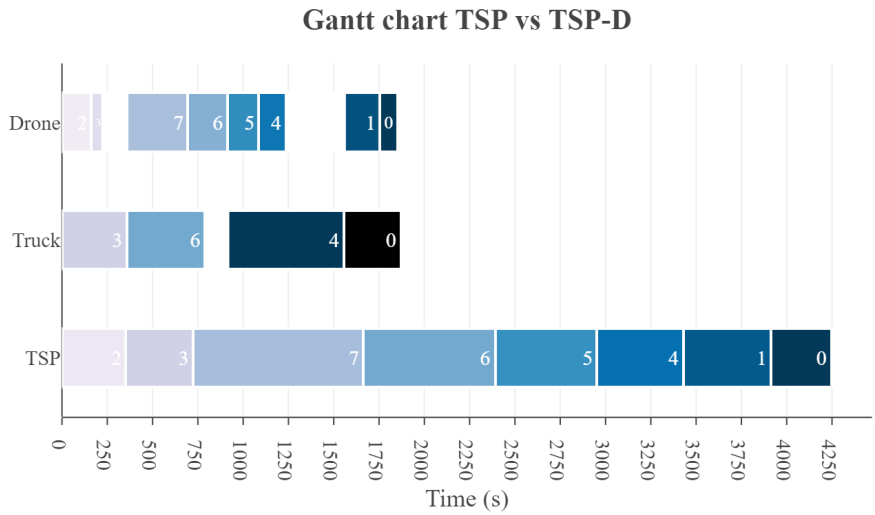


FIGURE 4.9: The Gantt chart showing the operational times in the TSP solution versus TSP-D solution.

4.4 Multiple Drones Implementation

An interesting extension of the previously discussed problem with a single truck and a single drone is made by the so called mFSTSP, in which one truck and multiple UAVs are available. This problem is less documented in literature, thus it is worth to try to extend the algorithms described above in this new scenario. Therefore, the available theory and algorithms developed will be exploited in order to find a solution where multiple drones can be launched from the truck. In particular, the approach will be focused on two UAVs implementation. However, this could be extended to multiple UAVs. Even though the methodology is based on a framework specifically designed for the TSP-D, the improvement that can be obtained with multiple drones still seemed worthy for a further analysis.

4.4.1 Local Search Algorithm - Multiple UAVs Adaptation

First of all, it has been necessary to modify the *Local Search Algorithm* so that it will be able to find a mFSTSP solution, possibly converging to at least a local minimum.

Note, however, that this aspect has not been emphasized at this stage because it was already clear that this algorithm would have been then inserted in the *HGA* framework, where it operates as an *education* algorithm and to eventually restore solutions that are not feasible.
The method implemented is shown in Algorithm 16.

Algorithm 16: Local search function for multiple drones

Initialize: tsp tour, response, truck only nodes, $\epsilon, s_R, s_L, \delta_t, \nu$
`max savings` = 0;
`C_prime` = C/`truck only nodes`;
`truck sub routes` = [`[tsp tour]` as many ν are available];
i^* = [[] as many ν are available];
j^* = [[] as many ν are available];
k^* = [[] as many ν are available];
`sub route uav flag` = [[] as many ν are available];
while *no improvement is produced* **do**
  for $n \in \nu$ **do**
    for `j` $\in$ `C_prime` **do**
      Call `savings`(*j, t, tsp tour*) function;
      for ***all*** `sub route` $\in$ `truck sub routes`[n] **do**
        if *this sub route is already assigned to a UAV* **then**
          Call `cost truck`(*j, t, savings, truck sub-routes, sub-route*) function;
        else
          Call `cost uav`(*j, t, savings, sub-route*) function;
        end
      end
    end
    if `max savings` > 0 **then**
      Call `perform update`(*i*[n], j*[n], k*[n], served by uav[n], truck sub-routes, drone nodes*) function;
      Reset `max savings`
    else
      STOP;
    end
  end
end

The first thing to be noticed is that the TSP solver has been removed, because it has been supposed that the `tsp tour` is an external input, coming from the *HGA* algorithm. For testing purposes, the `tsp tour` can be initialized as a random vector comprehending all customers that must be served.
Another peculiar aspect is that the algorithm requires the `truck sub-routes` to be initialized as the `tsp tour`, as it happened for the single drone single truck problem, but in this case for as many drones are available to be launched from the truck. In this way, each UAV has assigned its own schedule and it becomes simpler to detect their activities. In the same way i^*, j^*, k^* needs to be initialized as empty sets of as many elements as the number of drones available. In this way, ν different assignments per iteration can be done, where ν depicts the number of drones available.
Then, the main structure of the algorithm remains unchanged. However, an additional ***for*** *loop* appeared, in order to iterate the procedure for each drone available.

For example, in the case of $v = 2$, in the first cycle, i.e. when $n = 1$, the algorithm assigns a delivery task to UAV_1, that leads to the greatest value of `max savings`]. In the second iteration, with $n = 2$, it assigns to UAV_2 the travel that leads again to the greatest value of `max savings`. Then, if other possible improvements can be done, the algorithm starts back as in the TSP-D case.
The functions `savings` and `cost uav` required some modifications regarding the calculation of the `t` vector representing the arrival times of the truck at each served node. In particular, this aspect has become even more critical with respect to the TSP-D scenario since synchronization with multiple drones is harder to deal with. In any case, the input provided corresponds to a single truck-single drone problem, thus the general structure does not change. Indeed, the external loop ***for*** $n \in n_D$ allowed to separate each UAV activity and so it has been possible to treat each drone assignment as a TSP-D problem. However, the `cost truck` and `perform update` functions required a particular attention. The `cost truck` function is reported in Algorithm 17.

Algorithm 17: Calculate cost truck function for the mFSTSP problem

Input: `j, t, savings, truck sub-routes, sub-route`
find `a`, first node of the `sub-route`;
find `b`, first node of the `sub-route`;
for ***all*** `i` $\in$ `sub-route` **do**
  find `k`, node right after `i` in the `sub-route`;
  `cost` = $\tau_{ij} + \delta_t + \tau_{jk} - \tau_{ik}$;
  if `cost` < `savings` **then**
    if $t[b] - t[a] +$ `cost` $\leq \epsilon$ **then**
      if `savings - cost` > `max savings` **then**
        for *other* `truck sub-routes` *assigned to the other UAVs* **do**
          find a_D, first node of the `sub-route`;
          find b_D, first node of the `sub-route`;
          if $t[b_D] - t[a_D] +$ `cost` $\leq \epsilon$ **then**
            the truck node swap is feasible also considering other UAVs assignments;
          end
        end
        if *swap feasible for all UAVs assignment* **then**
          `served by uav = False;`
          $i^* =$ `i;`
          $j^* =$ `j;`
          $k^* =$ `k;`
          `max savings = savings - cost;`
        end
      end
    end
  end
end
return i^*, j^*, k^*, `served by uav, max savings`

The modification reported has been necessary since, if a truck node swap is feasible for a UAV assignment, it might be in contrast with other UAVs already assigned, hindering to reach feasibility for all drone travels.

Take as example Figure 4.10. The UAV_1 row is referred to the `truck sub-routes`

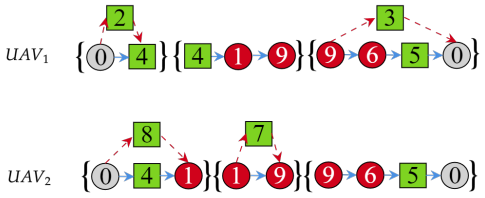


FIGURE 4.10: An example on how the `cost truck` function works for the case where $v = 2$.

and drone assignments made for the first drone available, similarly UAV_2 represents the same items for the second drone available. Note that, in any case, the two `truck sub-routes` must give rise to the same `tsp tour` and this is achieved by properly updating the `truck sub-routes` as it will be explained in the followings.
In this scenario, the only node still available for the assignment is customer *5*. This can be either assigned to UAV_2 or it can be swapped in the `tsp tour`. Indeed, in the first iteration, node *5* can not be assigned to UAV_1, since this is already travelling towards customer *3*. Thus, a truck node swap is investigated. Suppose that a convenient swap that leads to `max savings` > 0 is found by inserting node *5* between the nodes *0* and *4*. The feasibility of this swap for the `drone sortie (i, j, k) = (0, 2, 4)` is checked first. If it results as feasible, this does not mean that it would be feasible for other UAVs already assigned. Indeed, if the insertion of *5* within *0* and *4* would hinder the UAV_2 travel from *0* to *8* returning to *1*, then the truck swap must not be performed. If, instead, the `sub-route = [0, 5, 4, 1]` associated with the `drone sortie = (i, j, k) = (0, 8, 1)` still does not violate the feasibility constraints, then the truck swap can be feasibly made.
Finally, the `perform update` function is called, in order to update the `truck sub-routes` associated to the UAV_n considered, corresponding to the index n, but it is necessary to update the other `truck sub-routes` as well, according to the assignment performed.
The function is reported in Algorithm 18.
The `perform update` function resembles the previous version. Nonetheless, it needed minor tweaks in order to update the multiple `truck sub-routes` and the corresponding UAV assignments that are present in this scenario. In particular, it is possible to see that until Line 9 the structure remains unchanged. In Line 11, the UAV_n assignment is removed from the other `sub-route` associated to other UAVs. Then, from Line 18 to Line 20, the truck node swap is performed as in the single truck-single drone case. However, it is then necessary to update all the other `truck sub-routes` as well, by swapping $j^*[n]$ either inside the same `sub-route`, or among different `sub-route`.
The `time update` function required some modifications in order to obtain synchronization between the truck and the different UAVs available. This has been done by evaluating the time separately for each `truck sub-routes` associated to the corresponding UAV, as explained in Algorithm 6, but then a parallel check between the time vectors associated to each `truck sub-routes` is necessary in order to establish

whether the deliveries are coordinated or not.

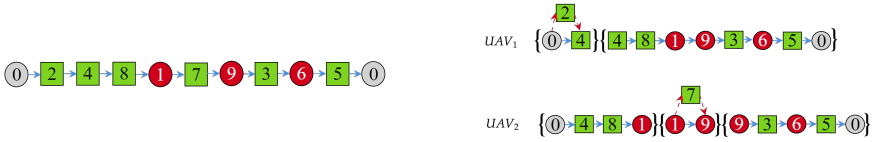
Algorithm 18: Perform update function for the mFSTSP problem

Input: $i^*[n], j^*[n], k^*[n]$, `served by uav[n]`, `truck sub-routes`, `drone nodes`

if `served by uav[n]` **then**
  remove $j^*[n]$ from the `tsp tour`;
  split the `truck sub-routes[n]`;
  update the `drone nodes[n]`;
  if $i^*[n] \in C'$ **then**
    remove $i^*[n]$ from C';
  end
  if $k^*[n] \in C'$ **then**
    remove $k^*[n]$ from C'
  end
  for ***all*** `truck sub-routes` **do**
    for `sub-route` *in each* `truck sub-routes` **do**
      remove $j^*[n]$ from `sub-route`;
    end
  end
  Call `time update(`*truck sub-routes, drone nodes*`)` function;
else
  remove $j^*[n]$ from the `tsp tour`;
  insert $j^*[n]$ in the new position in the `tsp tour`;
  update the `truck sub-routes[n]`;
  for ***all*** `truck sub-routes` **do**
    for `sub-route` *in each* `truck sub-routes` **do**
      make the truck node swap among the `sub-route`, according to where $j^*[n]$ is located;
    end
  end
  Call `time update(`*truck sub-routes, drone nodes*`)` function;
end
return `tsp tour, t, truck sub-routes`

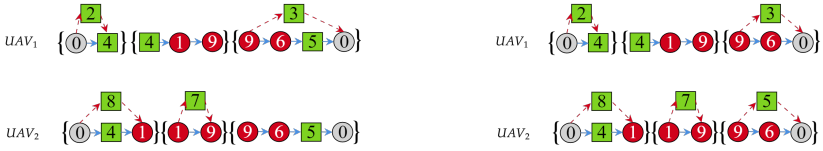
In the followings, a numerical example will be provided in order to better clarify how the algorithm works.

The `tsp tour` shown in Figure 4.11a is taken as input for the algorithm. This can be either produced by a solver, or it can be a random vector, or it can come from the general *HGA* framework, in which this algorithm will be inserted in. Then, the *truck sub-routes* will be initialized as `truck sub-routes = [[0,2,4,8,1,7,9,3,6,5,0], [0,2,4,8,1,7,9,3,6,5,0]]`, since $\nu = 2$. In this way, each drone will have its own `truck sub-routes` assigned to separate the travels that they can perform. However, as already mentioned, once an assignment is made, all the `truck sub-routes` need to be accordingly updated. At this stage, all the other relevant variables will be initialized, as depicted in Algorithm 16.
Next, the first iteration `for` $n \in \nu = [1,2]$ begins, i.e. $n = 1$. In this iteration, only the UAV_1 with its associated `truck sub-routes[1]` will be considered, as if it was a single drone, single truck scenario. The algorithm found that the most convenient drone assignment corresponds to the `uav sortie = (0,2,4)`. Therefore, the assignment will be stored, the `tsp tour` and `truck sub-routes` will be updated, but most importantly, from the *perform update* function the nodes *2* and *4* will be removed from



(A) The `tsp tour` input of the *Local Search* *function* adapted for two UAVs. Grey nodes represent the depot, green nodes represent drone eligible nodes and red nodes represent truck only nodes.

(B) After the 1^{st} iteration, the algorithm assigned node *2* to UAV_1 and node *7* to UAV_2.



(C) The 2^{nd} iteration produced a further assignment for UAV_1 travelling towards customer *3* and UAV_2 heading to customer *8*.

(D) Finally, the last drone eligible node that could have been assigned *5* is served by UAV_2.

FIGURE 4.11: Example of how algorithm works.

the set of drone eligible nodes, C'. This is done to avoid that the same assignment will occur also for the following drones available and to avoid that customer *4* will be removed from the `tsp tour`. The latter condition shall be prevented since the `uav sortie = (0,2,4)` forecasts that UAV_1 will be recovered by the truck at node *4*, thus the truck needs to visit that customer.
In the second iteration of the ***for*** *loop*, i.e. when $n = 2$, the UAV_2 and related `truck sub-routes[2] = [0,4,8,1,7,9,3,6,5,0]` are considered. Note that node *2* has been removed from the `truck sub-route[2]` since it has been assigned to UAV_1. For this drone, it has been found that the most convenient assignment is the `uav sortie = (1,7,9)`, therefore the `tsp tour`, all the `truck sub-routes` and C' will be updated accordingly. The final situation after the first improvement cycle is shown in Figure 4.11b.
Then, the algorithm goes out the ***for*** *loop*, but since in the first iteration has produced some improvements in the overall solution, it starts back trying to find new assignments in order to reduce the overall makespan of the delivery process. In this iteration, the algorithm has found the following assignment: `uav sortie = (9,3,0)` for UAV_1 and `uav sortie = (0,8,1)` for UAV_2. This is depicted in Figure 4.11c.
The algorithm keeps going another time, where only two possibilities are still present:

- make a truck node swap for node *5* for UAV_1;
- make a drone assignment for node *5* for UAV_2;

Customer *4* can not be assigned to any drone or swapped in the `tsp tour` because it constitutes a drone retrieval point for UAV_1 and UAV_2 is already occupied in the meantime, travelling towards customer *8*. Thus, a tentative truck node swap for customer *5* is performed first for UAV_1. This happens because UAV_1 is already busy, carrying its parcel to customer *3*. Supposing that there are not truck node swaps that allows feasible and/or improved solutions (it is possible to refer to the

explanation of Figure 4.10 to have a better insight of this particular problem), the algorithm will then move on to UAV_2 and it tries to allocate it to a drone travel. This has been found as a feasible solution so the `uav sortie = (9,5,0)` is assigned, as it shown in Figure 4.11d.
Finally, the algorithm will stop because no other improvements can be done, but in this particular case, it also stops because the C' set has become empty at this stage.

4.4.2 Hybrid Genetic Algorithm - Multiple UAVs Adaptation

The *HGA* framework has been modified as well in both of its versions, the *Only Feasible Solutions Variant* and the *Infeasible Solutions Variant*. The modifications introduced in order to deal with multiple drones has narrowed the difference between these two variants, as it will be discussed in the next Section.

The *Only Feasible* Solutions Variant

First, the *Only Feasible Variant* will be discussed. Basically, the main structure of the algorithm remains unchanged. The only exception is that in Line 7 there is already present the `split` function. This insertion has been necessary since the *Local Search function* for multiple drones has revealed quite inefficient in finding a local minimum for a general case that starts from a random `tsp tour`. Therefore, in this case, the `split` function finds a tentative solution pseudo-randomly, that can be possibly improved by the *Local Search function*, refining locally the provided tentative solution for the mFSTSP problem. However, the main difference with the `split` function described in Section 4.3.2 is that, in this case, the `split` function must return *Only Feasible* solutions for ν drones.

Algorithm 19: HGA Only Feasible solution Variant for the mFSTSP problem

Initialize: response, truck only nodes, $\epsilon, s_R, s_L, \delta_t, \nu$
1 **generate** random population of n individuals;
2 $\gamma = 0$
3 **while** $\gamma <$ `threshold` **do**
4 $\gamma = \gamma + 1$;
5 Call `parents selection`(*population, fit values*) function;
6 Call `child generation`(*parents, drone deliveries*) function;
7 Call `split`(*child*) function;
8 Call `local search`(*split child*) function;
9 Call `restore`(*educated child, educated drone deliveries*) function;
10 Call `select survivors`(*restored child, educated child, educated drone deliveries, population*) function;
11 **end**
12 Call `select fittest individual`(*population, fit values*) function;

Since all the other functions are designed in the exact same way as in the *HGA* for the TSP-D problem, they will not be described here. A particular emphasis will be devoted to the `split` function instead, which is the main new feature for the multiple drones adaption of the *HGA* algorithm.

The `split` function is shown in Algorithm 20.
This has nested inside few more functions that will be described in the followings.

Algorithm 20: Split function for the mFSTSP problem

Input: `child`
select `number of drone nodes` n_d as in Equation 4.2;
sample n_d random `drone nodes` $\in C'$;
remove assigned `drone nodes` from `child`;
calculate `t` vector of `child`;
Call `create uav sorties(`*child, drone nodes,* n_d*, t*`)` function;
while `infeasible customer` *is* ***not*** *empty* **do**
  Call `insert infeasible customers(`*child, infeasible customer*`)`;
  reset assigned `drone nodes`;
  update `t`;
  Call `create uav sorties(`*child, drone nodes,* n_d*, t*`)` function;
end
reset `drone nodes` and C';
Call `truck sub-routes update(`*child, uav sorties,* ν`)` function;
Call `time update(`*truck sub-routes, drone nodes*`)` function;
return `split child, t, split truck sub-routes, split drone deliveries`

First, it is necessary to define the number of customers that can be served by ν drones, n_d. In this case, it has been chosen that they should always be the maximum possible. The formula to calculate n_d is shown in Equation 4.2:

$$\begin{aligned} & n_d = \texttt{len}(C) - \left\lceil \frac{C-\nu}{\nu+1} \right\rceil \\ & if\ n_d > \texttt{len}(C') : n_d = \texttt{len}(C') \end{aligned} \tag{4.2}$$

The quantity $\lceil \frac{C-\nu}{\nu+1} \rceil$ has been taken from the [Murray & Raj, 2019] [14] and it is called *lower truck limit*, or *LTL*. This corresponds to the minimum number of customers that can be served by the truck, in presence of ν drones. C depicts the total number of customers that must be served.
Then, n_d `drone nodes` will be sampled from the drone eligible nodes set, C'. These nodes will be removed from the `child` vector. Note that in this case, opposite to the `split` function shown in Section 4.3.2, it is not necessary to remove consecutive sampled nodes as they appear in the `child` vector, since now there are multiple drones available that can serve these customers.
Then the `create uav sorties`, shown in Algorithm 21, is called in order to assign all the `drone nodes`. However, some of them might result to be infeasible, thus they need to be re-inserted in the `child` vector as a truck node. This is done by the `insert infeasible customers` function, depicted in Algorithm 22.
The ***while*** *loop* is used because the `insert infeasible customers` function does not deal with drone feasibility constraints, but it only accounts for the best insertion of the infeasible customer inside the current `tsp tour` represented by the `child` vector. Thus, until `infeasible customers` are produced, the algorithm tries to assign less `drone nodes` to a `uav sortie` and it inserts them in the `tsp tour` in the most convenient way.
Once a feasible solution has been found, the algorithm exits from the ***while*** *loop* and it produces a mFSTSP solution in a format that will be readable by the *Local Search function*, i.e. by creating the `truck sub-routes`, `drone nodes` and calculating the

corresponding `t` vector.

Algorithm 21: Create UAV sortie function for the mFSTSP problem

```
   Input: child, drone nodes, n_d, t
 1 initialize number of options as vector of zeros, with as many nodes as in
     tsp tour;
 2 for n ∈ range(n_d) do
 3     for j ∈ drone nodes do
 4         for i ∈ tsp tour do
 5             find k node in the tsp tour after i;
 6             if τ'_ij + δ_t + τ'_jk < ϵ and τ_ik < ϵ then
 7                 number options[j] = number options[j] + 1
 8             end
 9         end
10     end
11 end
12 while there are unassigned customers do
13     initialize wait time = ∞;
14     pick j as node with less number options;
15     if number option[j] = 0 then
16         insert j in infeasible customers;
17     else
18         j = j';
19         for i ∈ tsp tour do
20             for n ∈ available uav[i] do
21                 find k node in the tsp tour after i;
22                 if τ'_ij + δ_t + τ'_jk < ϵ and τ_ik < ϵ then
23                     w = τ'_ij + δ_t + τ'_jk − t[k] - t[i] if wait time ≥ 0 and w <
                        wait time or wait time < w < 0 then
24                         wait time = w;
25                         n* = n;
26                         i* = i;
27                         j* = j';
28                         k* = k;
29                     end
30                 end
31             end
32         end
33         if wait time = ∞ then
34             insert j' in infeasible customers;
35         else
36             assign uav sortie = (n*,i*,j*,k*)
37         end
38     end
39 end
40 return uav sorties, infeasible customers
```

In the `create uav sortie` function, from Line 2 to Line 11 the `number options` available for each `drone nodes` that needs to be assigned is calculated. In particular, the `number options` vector denotes the number of feasible drone assignments for node `j` that can be done considering `i` and `k` as consecutive nodes in the `tsp tour`.

`i` and `k` are considered as consecutive nodes in the `tsp tour` since it has been supposed that the least amount of customers shall be served by the truck. Therefore, the drone travels must happen between consecutive nodes. Then, if some drone assignment will result as infeasible, the `insert infeasible customers` function will be entitled of inserting additional truck nodes that can not be assigned to `uav sortie` and if possible the *Local Search function* will improve the overall mFSTSP solution.
From Line 12 to Line 39 the `drone nodes` are progressively assigned to `uav sorties`. If a node does not have any option to be feasibly served by a UAV, it will be inserted in the `infeasible customers` set. Instead, if `number options` associated to the considered node `j` is greater than *0*, the nodes that has less `number options` is assigned first preferably, because fewer possibilities are available and so it is advisable to address these customers first, in order to serve the maximum number of customers with UAVs. Indeed, it may happen that previous assignments made all the UAVs busy, decreasing the corresponding `number options`. For this reason, it is better to leave the `drone nodes` with more options at the end of the assignment.

Algorithm 22: Insert infeasible customers function for the mFSTSP problem

Input: `infeasible customers`, `tsp tour`
1 **while** *there are* `infeasible customers` **do**
2 **for** `i` ∈ `infeasible customers` **do**
3 **for** `k` ∈ `tsp tour` **do**
4 **find** `p`: position of `k` in `tsp tour`;
5 **find** `h` node right before `k`;
6 $(p^*, i^*) = \underset{(p,i) \in TSPtour}{\arg min} \ (\tau_{hi} + \delta_t + \tau_{ik} - \tau_{hk})$;
7 **end**
8 **end**
9 **insert** i^* in the `tsp tour` at position p^*;
10 **end**
11 **return** `tsp tour`

Algorithm 23: Truck sub-routes update function for the mFSTSP problem

Input: `child`, `uav sorties`, ν
1 **for** $n \in$ `range`(n_d) **do**
2 **for** ***each*** `uav sortie` **do**
3 `uav sortie = (n,i,j,k));`
4 **remove** `i, j, k` from C', if present;
5 **for** `sub-route` ∈ `truck sub-routes[n]` **do**
6 **remove** `j` from the `sub-route`;
7 **split** the `truck sub-routes`;
8 **update** the drone delivery nodes;
9 **BREAK**;
10 **end**
11 **end**
12 **end**
13 **return** `truck sub routes, drone nodes`

The `insert infeasible customers` function simply inserts all the `infeasible customers` inside the `tsp tour` that will lead to the minimum impact on the final mFSTSP solution.
Once all the nodes have been assigned either to the UAVs or the truck, the solution must be updated according to the `truck sub-routes update` function.

Finally, the `fittest individual` with its associated solution is selected as in the single truck-single drone scenario.

The *Infeasible* Solutions Variant

The *Infeasible* Solutions Variant of the *HGA* for multiple drones shares the same structure of the Algorithm 12 described for a single truck and a single drone. Furthermore, it shares many features of the *Only Feasible Variant* as well, since this has already built in a `split` function. In this case, however, the `split` function does not necessarily have to produce a *feasible* solution, thus the ***while*** *loop* that guaranteed *feasible* mFSTSP solutions in Algorithm 20 can be ignored.
The new `split` function is depicted in Algorithm 24.

Algorithm 24: Split function of Infeasible Solutions Variant for the mFSTSP problem

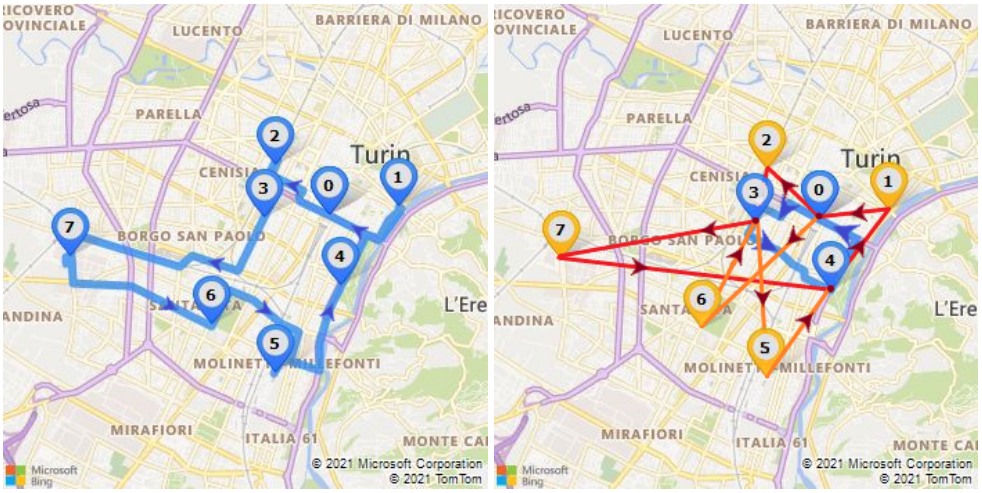
Input: `child`
select `number of drone nodes` n_d as in Equation 4.2;
sample n_d random `drone nodes` $\in C'$;
if *number of infeasible solutions* $= \eta$ ***or*** `increment variable` $= 0$ **then**
    `increment variable` = `increment variable` + 1;
    if `increment variable` $= \frac{n_d}{2}$ **then**
        `increment variable` = 0;
        $n_d = 1$;
    end
end
$n_d = n_d -$ `increment variable`;
remove assigned `drone nodes` from `child`;
calculate `t` vector of `child`;
Call `create uav sorties`(*child, drone nodes,* n_d*,* t) function;
Call `truck sub-routes update`(*child, uav sorties,* v) function;
Call `time update`(*truck sub-routes, drone nodes*) function;
return `split child, t, split truck sub-routes, split drone deliveries`

Another interesting feature of this function consists in the introduction of the so called `increment variable`. This is used to reduce the number of nodes arbitrarily assigned to UAVs if many infeasible solutions are produced. Indeed, as supposed in Algorithm 20, the seeded `drone nodes` shall always be the maximum, in order to reduce the customers served by the truck to minimize the makespan of the delivery process. However, in the *Only Feasible Variant*, if some of the customers have no possibilities to be served by UAVs, because, for example, they are located too far from the truck route to be feasibly served by a UAV that has a fixed endurance limit, then, they can be re-inserted in the `tsp tour` by the `insert infeasible customer` function. Instead, in this case, if the customer's location pattern is such that many nodes are located far from each other and so they have fewer options to be served by UAVs during the truck route, if the maximum number of customers is always assigned to the UAVs, the `split` function will produce only infeasible solutions. Thus, the `increment variable` is used to decrease the the number of customers served by drones, n_d, if many infeasible solutions have been produced previously. In this way, it is more likely that the `split` function will eventually produce a pseudo-random feasible solution. Once the `increment variable` reaches a certain threshold it is reset, in order to always tentatively assign some customers to the UAVs.

The `create uav sorties` function called inside the `split` function, is slightly different from the previous version, since now the `number options` are not considered anymore, but all the customers are assigned to a `uav sortie`, regardless of their feasibility. Therefore, in this case, the `create uav sorties` function will not return any `infeasible customer`.

4.4.3 The Real Case Scenario Example solved with *HGA Infeasible Variant - Multiple Drones Adaptation*

To see how the algorithm adapted for multiple UAVs performs, the example of the real case scenario already discussed in Section 4.2 is provided in the followings. The inputs are always given by Tables 4.1a, 4.1b and 4.1c.



(A) The conventional TSP solution of the instance considered.

(B) The mFSTSP solution produced by the *Hybrid Genetic Algorithm Infeasible Variant - Multple Drones Adaptation*.

FIGURE 4.12: Example of the solution produced by the algorithm in a real case scenario where seven customers need to be served. In this instance every node has been considered as eligible for drone delivery.

From Figure 4.12b it is possible to notice that the customers served by the truck are reduced to the minimum number possible for $\nu = 2$, according to Equation 4.2. This has been possible since, in this instance, all the customers are located within reachable distance for the endurance limits of the UAVs. UAV_1 travels are depicted in the Figure by the *red* lines, while UAV_2 assignments are represented by *orange* lines.
From Figure 4.13, it is possible to see that the completion time of the overall delivery process is 1390*s*, that corresponds to an important improvement with respect to the TSP-D scenario, as expected. This advantage is already evident even for this example, where a small number of customers are present. A further analysis and comparison among the different algorithms will be provided in Chapter 5.

In Figure 4.13 it is possible to see the truck operations, denoted by *Truck* row, the first UAV deliveries, denoted by $Drone_1$ and the second UAV deliveries, shown in row $Drone_2$.
The truck and both the drones starts from the depot *0*. The truck heads toward customer *3*, where both UAVs will be retrieved again. $Drone_1$ goes to customer *2* and comes back at node *3* before the truck, thus it needs to wait, hovering in place.

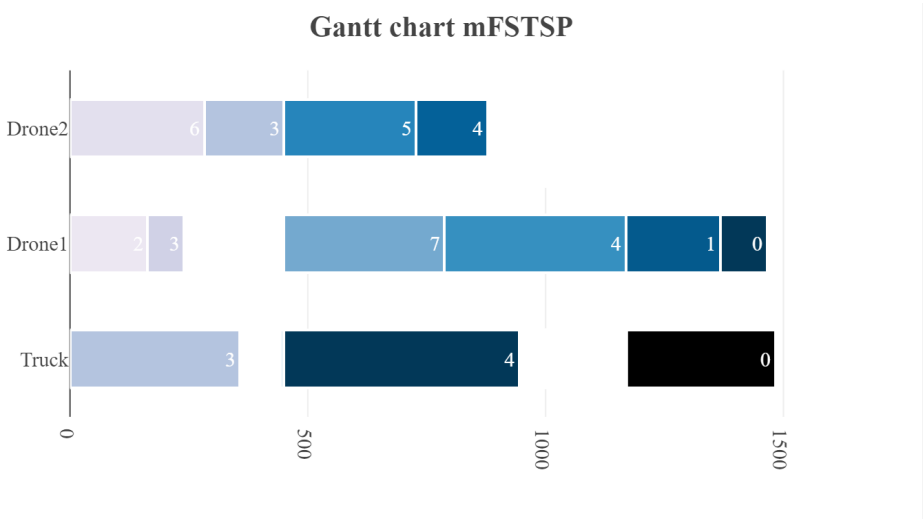


FIGURE 4.13: The Gantt chart showing the operational times in the mFSTSP solution, with $\nu = 2$.

$Drone_2$ delivers its parcel to node *6* and it comes back to the truck route at node *3* being the last vehicle to arrive at that node. Thus, before heading to the next customer, the truck must wait for $Drone_2$. Once all the drones are retrieved and the maintenance has been done, all the vehicles heads towards the next delivery. In particular, the truck goes to customer *4*, while $Drone_1$ goes to node *7* and $Drone_2$ goes to customer *5*. When $Drone_2$ arrives first at node *4*, it has to wait for the truck to be retrieved. Then, the truck needs to wait for $Drone_1$, that is busy in the longest travel for this `sub-route`. Finally, $Drone_1$ heads to customer *1* while the truck with on board $Drone_2$ comes back to the depot *0*.
The activities of all the vehicles are synchronized as supposed at the beginning and the solution provided an important improvement with respect to the initial TSP and TSP-D solutions as expected.

4.5 mFSTSP Exact Method - An Alternative Approach for Multiple Drones

In this Section an alternative approach to the mFSTSP solution will be discussed. Since this is based on the [Murray and Raj, 2019] [14] work, it has been used as a benchmark because this method represents the current state-of-the-art for solutions of this type of problem. However, the code implementation has been greatly simplified from the original method proposed, thus it should be considered as a sub-optimal solution to which the previously discussed algorithms shall be compared.

This method has been tested during the design phase of the other approaches presented in this Thesis, and it will be discussed in the followings.

Algorithm 25: Exact method algorithm, inspired by [Murray and Raj, 2019] [14] for the mFSTSP problem

Initialize: truck input, response, truck only nodes, ϵ, δ_t, ν
calculate LTL as in Equation 4.2;
`low bound` = ∞
while *LTL* < `len(truck input)` **do**
  `C_prime` = C/`truck only nodes`;
  initialize `tsp tour` = `[0,0]`;
  Call `cost truck calculation`(*tsp tour, truck input, C_prime, LTL*) function;
  Call `customer division`(*tsp tour, C_prime*);
  Call `time update`(*tsp tour*) function;
  Call `create uav sorties`(*tsp tour, uav customers, time,* ν);
  Call `insert infeasible customers`(*tsp tour*);
  Call `create truck sub-routes`(*tsp tour, drone nodes*) function;
  if *makespan* < `low bound and LTL` $\leq$ `len(truck customer)` ***and not*** *infeasible* **then**
    save the mFSTSP solution;
    update `low bound`;
    increase LTL;
  else
    increase LTL;
  end
end

This algorithm deals with the mFSTSP problem in a completely different way with respect to the *HGA* heuristic; indeed, this produces a *unique* solution, while the previously seen heuristics seek an optimal solution iteratively by exploring several possible combinations, which shall produce progressively improved solutions.
First of all, the algorithm computes the *LTL*, which corresponds to the minimum possible number of customers that can be served by the truck if ν drones are present, as discussed in Equation 4.2. In this way, the maximum number of customers possible will be served by drones. This is done in order to reduce the total makespan since the drones are supposed to be faster than the truck in a generic urban scenario.
Then, a ***while*** *loop* is initiated because, if for feasibility reasons the algorithm is not able to assign the maximum number of customers to the drones, this should be able to find anyway a feasible solution that will be obtained more likely if the customers assigned to the truck is increased. The set of drone eligible drones, C', is initialized every time the loop starts, as well as the `tsp tour`, that is defined initially just with the initial and final destinations that correspond to the same depot *0*.
At this point *LTL* customers are inserted in the `tsp tour` according to the `cost truck`

`calculation` function.

Algorithm 26: Calculate cost truck function for the mFSTSP problem

```
   Input: tsp tour, truck input, truck only nodes, LTL
 1 while index < LTL do
 2     min cost = ∞;
 3     if truck only nodes is not empty then
 4         for j ∈ truck only nodes do
 5             for i ∈ tsp tour do
 6                 find k, node right after i in tsp tour;
 7                 cost = τ_ij + δ_t + τ_jk + −τ_ik;
 8                 if cost < min cost then
 9                     j star = j;
10                     k star = k;
11                     min cost = cost
12                 end
13             end
14         end
15         insert j star before k star;
16         index = index + 1;
17     else
18         for j ∈ truck input do
19             for i ∈ tsp tour do
20                 find k, node right after i in tsp tour;
21                 cost = τ_ij + δ_t + τ_jk + −τ_ik;
22                 if cost < min cost then
23                     j star = j;
24                     k star = k;
25                     min cost = cost
26                 end
27             end
28         end
29         insert j star before k star;
30         index = index + 1;
31     end
32 end
33 return tsp tour
```

This function inserts the requested number of customers *LTL* such that, among all the customers available, the `tsp tour` would be the shortest possible. Therefore, it solves a problem similar to a conventional TSP, where the truck should not serve all the customers, but only *LTL* customers that are the most convenient ones. Note that, if some customers are not eligible for drone delivery, for example for capacity constraints, these customers shall be preferably assigned first, as it is done from Line 3 to Line 17, since they can not be addressed by drones in any way. Otherwise, the customers are selected and conveniently inserted from the `truck input`, as it can be seen from Line 18 to Line 31.

For example, if a set of customer `truck input = [1:10]` is present, the `truck only nodes = [2, 5, 8]` and $LTL = 5$, then the `cost truck calculation` must find the best TSP tour composed by *5* customers belonging to the `truck input` set, which could be `tsp tour = [0, 3, 8, 2, 5, 10, 0]`, where the insertion of *3* and *10* and

their disposition has been found as the most convenient possible. Consequently, the remaining customers, called `uav customers = [1, 4, 6, 7, 9]` will be assigned to UAVs automatically. This division among customers served by truck and UAVs is dealt by the `customer division` function.
Subsequently, the algorithm tentatively assign a mFSTSP solution according to the `tsp tour` and `uav customers` available, by calling the `create uav sorties`, `insert infeasible customers` and `create truck sub-routes` functions. These have exactly the same structure as discussed respectively in Algorithms 21, 22 and 23. If the solution's makespan is lower than the `low bound`, initialized as ∞, then the solution is saved, but the algorithm seeks for another possible solution by increasing the *LTL* and repeating the procedure again. This is useful when nodes are located further apart and some of them can not feasibly served by UAVs, thus, the first attempts are either infeasible or poor solutions due to the re-insertion of customers made by the `insert infeasible customers` function, that integrate on the `tsp tour` the customers that can not feasibly served by the UAVs.
In the next Chapter, an insight of the performance of the algorithms developed will be discussed with different scenarios featuring randomly sampled customers in a given area, in order to investigate better the behavior of the methodologies explained above.

Chapter 5

Results

In this Chapter, a numerical analysis has been carried out in order to test the algorithms presented previously. These experiments allowed to acquire a better insight on performances and to demonstrate the behavior of the approaches that has been discussed theoretically.
The computational work has been carried out with a laptop ASUS Vivobook F510U equipped with a quad-core Intel ® Core ™ i5-8250U 1.6 GHz and 8 GB of RAM installed.
The codes have been developed in PyCharm using Python 3.9.
The parameters characterizing UAVs operations are shown in Table 5.1. These parameters are necessary in order to initialize any algorithm.

Parameters				
ϵ	s_R	s_L	Avg Drone Speed	Truck Only Nodes
25 min	1 min	1 min	40 km/h	-

TABLE 5.1: Test parameters.

The assumptions on these values are motivated as follows:

- *drone endurance limit*, ϵ, has been set to 25 minutes because, even if this value does not stretch to the values that can be reached by the current state-of-the-art of drones technology, battery life during parcel deliveries may be hindered by the weight of the payload and by the varying relative speed between the drone and the air flow. However, these effects have not been accounted for, thus a constant battery life model with a conservative value has been selected;
- *service launch and service recovery times*, s_R and s_L, have been supposed to last 1 minute each, in order to account for setting the drone for take off or landing and possibly for battery swap;
- *Average Drone Speed*, has been set to 40 km/h. This value has been considered reasonable because even though it is lower than the maximum speeds achievable by drones for parcel delivery, landing and take off maneuvers must be taken into account as they occur at a lower speed than cruise speed. Furthermore, as for battery life, wind and payload might adversely affect this parameter as well. However, a constant speed model has been assumed in order to not make the model too much sophisticated;
- *Truck Only Nodes* have been assumed to be an empty set, i.e. all nodes have been declared eligible for drone delivery. This has been done in order to let the algorithms free to find the best possible solutions given the set of customers and the fleet for delivery available. However, the algorithms are able to deal with restrictions related to some parcels that must be served by the truck.

Another important parameter to set is the set of coordinates depicting the customers that must be served in the delivery process. For these experiments, they have been generated randomly via http://www.geomidpoint.com/random/ in a rectangular map comprehending the Turin urban area with the following limits:

- N limit 45.120371
- S limit 44.995261
- W limit 7.595798
- E limit 7.731985

	Coordinates	
0	45.09023778	7.86381856
1	45.00477682	7.81873701
2	44.99679575	7.54456076
3	45.13373049	7.6307671
4	45.05939138	7.7530238
5	45.05952062	7.70488864
6	45.10627511	7.63601916
7	45.15397837	7.80954089
8	45.04886232	7.66584172
9	45.03143891	7.69105793
10	45.092564	7.84682093

(A) Set of coordinates generated randomly via http://www.geomidpoint.com/random/ in a rectangular map comprehending the Turin urban area.

0	1049	1128	739	471	995	713	644	448	593	191
1030	0	772	1610	733	1192	1216	1160	1136	1536	1222
1179	829	0	1379	1059	1939	855	1625	1398	1621	1283
840	1501	1362	0	1098	1357	758	1009	957	424	728
404	849	925	1057	0	963	716	850	623	997	595
1006	1337	1908	1475	983	0	1448	774	751	1090	973
655	1040	848	839	674	1405	0	1084	857	1081	731
634	1132	1536	1231	611	507	1076	0	379	847	601
450	1195	1324	1000	541	652	892	288	0	629	350
696	1546	1509	458	975	1125	1040	828	649	0	551
373	1197	1310	775	543	929	896	574	381	533	0

(B) Truck travel time matrix, generated trough the Bing Maps API and taken as cost function matrix.

0	590	571	320	213	395	303	213	144	287	79
590	0	367	909	376	618	680	521	522	841	591
571	367	0	863	416	806	478	637	595	858	619
320	909	863	0	533	543	452	447	408	152	325
213	376	416	533	0	402	377	221	181	479	225
395	618	806	543	402	0	695	194	263	401	317
303	680	478	452	377	695	0	505	433	515	383
213	521	637	447	221	194	505	0	71	339	146
144	522	595	408	181	263	433	71	0	319	86
287	841	858	152	479	401	515	339	319	0	254
79	591	619	325	225	317	383	146	86	254	0

(C) Drone travel time matrix, generated trough the Euclidean distance matrix taken from the Bing Maps API and by assuming an average drone speed of 40 km/h.

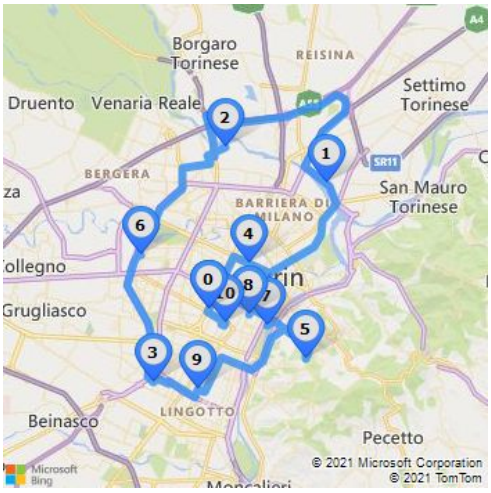
TABLE 5.2: Input data for the algorithm.

The random sampling has produced a set of *10* customers plus the *Depot* from which the truck must depart and go back to at the end of the delivery process.
The set of coordinates are shown in Table 5.2a.
The list of coordinates have been sent to the Bing Maps API REST Services in order

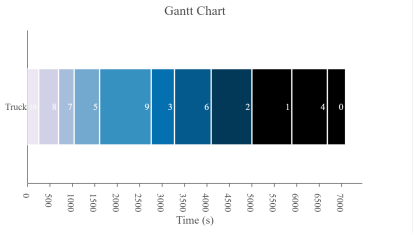
to obtain Table 5.2b, that is the truck travel time matrix, expressed in seconds, calculated according to road network available for the area considered. If called in real time, this will be affected by current traffic data as well. Table 5.2c is the drone travel time matrix. This has been calculated, instead, according to the Euclidean distances among coordinates that have been divided by the Average Drone Speed.
The parameters presented in Table 5.1 have been considered for all the algorithms and tests carried out in Sections 5.1, 5.2, 5.3 and 5.4.

5.1 Test 1 - Algorithms Comparison - Ten Customers Scenario

First, a comparison among algorithms operating on the defined scenario is discussed. In this case, graphical outputs has been produced, in order to clarify how the approaches operate.

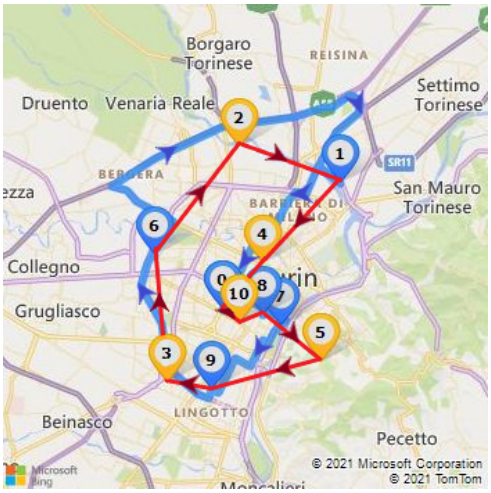


(A) The conventional TSP solution map.

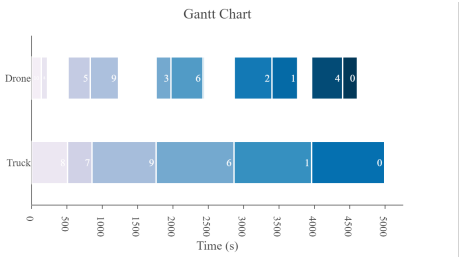


(B) The Gantt chart of the TSP solution.

FIGURE 5.1: The TSP solution for the instance considered, where ten customers must be served.

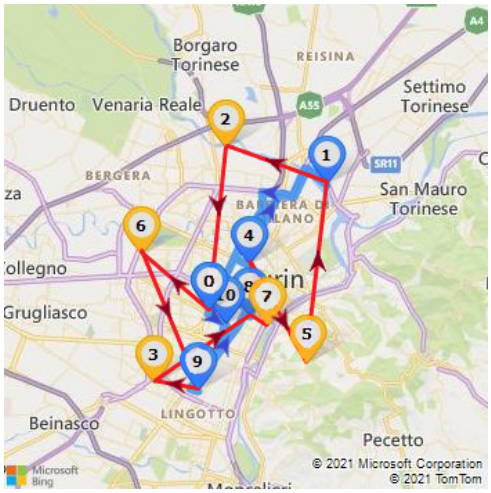


(A) The TSP-D solution produced by the *Local Search Algorithm*.

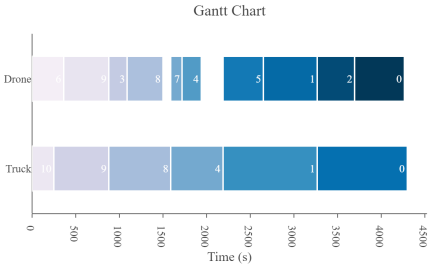


(B) The Gantt chart of the TSP-D solution.

FIGURE 5.2: TSP-D solution of *Local Search Algorithm*.

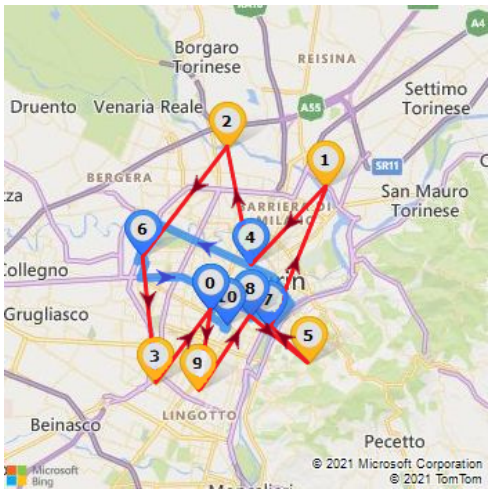


(A) The TSP-D solution produced by the *HGA - Only Feasible Variant*.

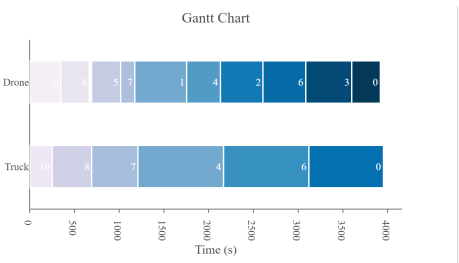


(B) The Gantt chart of the TSP-D solution.

FIGURE 5.3: TSP-D solution of *HGA - Only Feasible Variant* algorithm.

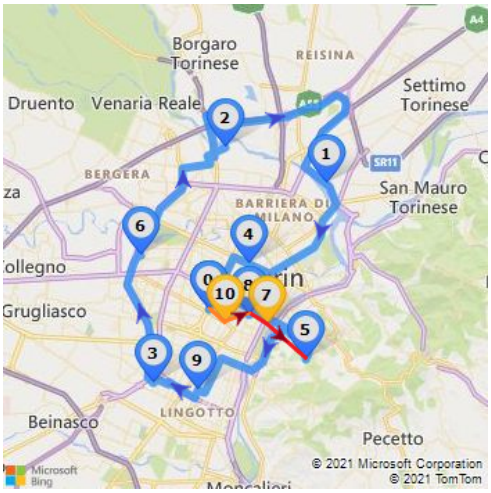


(A) The TSP-D solution produced by the *HGA - Infeasible Variant*.

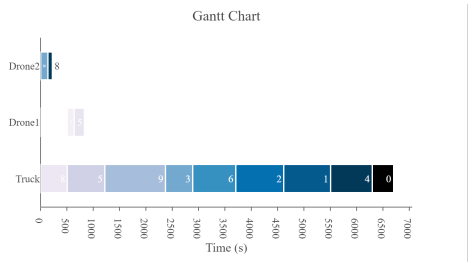


(B) The Gantt chart of the TSP-D solution.

FIGURE 5.4: TSP-D solution of *HGA - Infeasible Variant* algorithm.

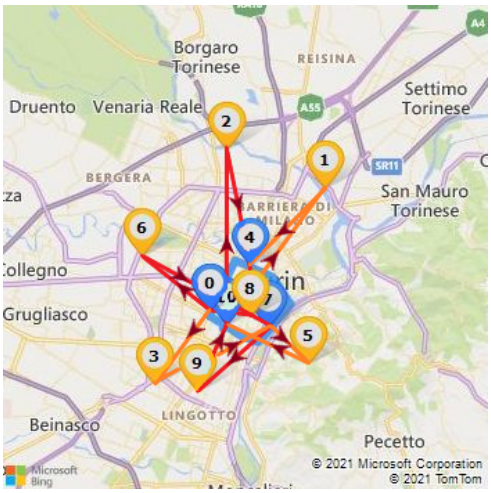


(A) The mFSTSP solution produced by the *Local Search Function - Multiple Drones*.

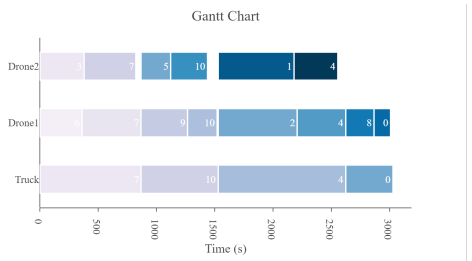


(B) The Gantt chart of the mFSTSP solution.

FIGURE 5.5: mFSTSP solution of *Local Search Algorithm - Multiple Drones*.

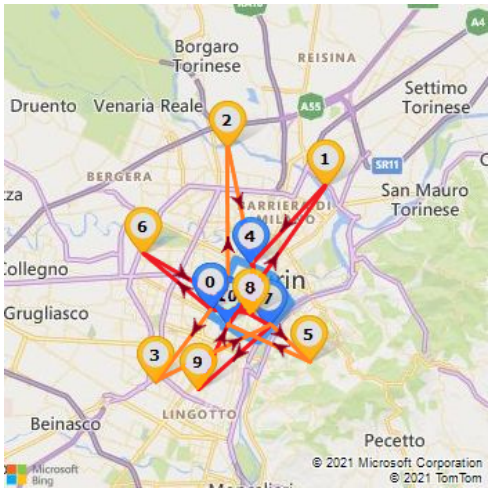


(A) The mFSTSP solution produced by the *HGA Only Feasible Variant - Multiple Drones*.

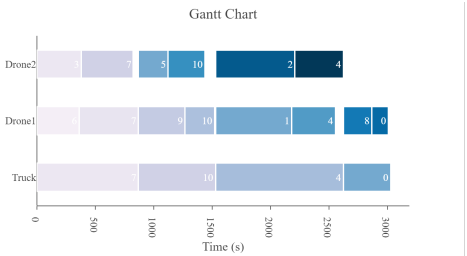


(B) The Gantt chart of the mFSTSP solution.

FIGURE 5.6: mFSTSP solution of *HGA Only Feasible Variant - Multiple Drones*.

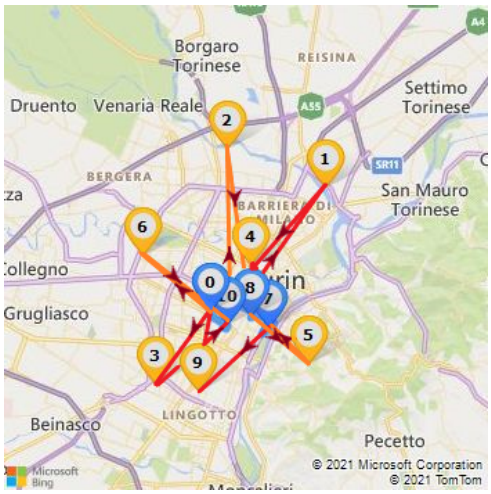


(A) The mFSTSP solution produced by the *HGA Infeasible Variant - Multiple Drones*.

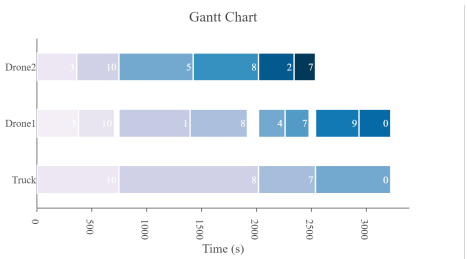


(B) The Gantt chart of the mFSTSP solution.

FIGURE 5.7: mFSTSP solution of *HGA Infeasible Variant - Multiple Drones*.



(A) The mFSTSP solution produced by the *Exact Method - Multiple Drones*.



(B) The Gantt chart of the mFSTSP solution.

FIGURE 5.8: mFSTSP solution of *Exact Method - Multiple Drones*.

The Figures above show the progression of the algorithms towards a better solution. Indeed, the total truck travel time is minimized, reducing the number of customers that must be visited by the truck, when one drone is available. In particular, the efficiency of the algorithm is depicted by the tightness of the schedule between drone deliveries and truck deliveries, that can be seen from the Gantt charts. This becomes more evident for the *HGA - Infeasible Variant* algorithm in Figure 5.4, where almost no waiting times between the truck and the drone are occurring during the delivery process.
Advantages are emphasized if two drones are available, as expected. The enlargement of the drone fleet has produced a reduced truck route, without undermining coordination, that is always obtained. Indeed, multiple drones are able to serve multiple customers simultaneously while the truck is able to reduce its delivery schedule.

In Table 5.3, results are reported in terms of the optimized variable of the test for a single iteration.

	Completion Time			
	TSP	1 UAV	2 UAVs	*Savings over TSP (%)*
TSP solver	1 h 58 min and 7 s	-	-	-
Local Search	-	1 h 23 min and 15 s	-	34.9
HGA - Feasible	-	1 h 11 min and 38 s	-	39.8
HGA - Infeasible	-	1 h 5 min and 52 s	-	44.6
Local Search - mUAVs	-	-	1 h 51 min and 40 s	5.5
HGA Feasible - mUAVs	-	-	50 min and 28 s	57.3
HGA Infeasible - mUAVs	-	-	50 min and 28 s	57.3
Exact Method	-	-	53 min and 43 s	54.5

TABLE 5.3: The results in terms of the optimized variable, the *completion time* of the delivery process, of the various algorithms that has been tested.

Also in this Table, the progressive enhancement of the solution is highlighted, with the only exception that is constituted by the *Local Search Algorithm - Multiple Drones*. This, however, has been used just as an intermediate step to build the optimized algorithms *HGA Feasible - Multiple Drones* and *HGA Only Infeasible - Multiple Drones*, thus it has been ignored.

5.2 Test 2 - Monte Carlo Simulation for Heuristic Algorithms

In this Section, several simulation has been performed using the *Test 1* scenario, in order to obtain a set of data that helps visualizing the results and establishing the efficiency of the algorithms implemented. The solutions will be distributed according to a probability function, so that possible future outcomes of the algorithms can be predicted with more ease.
The results are shown in Figure 5.9, through a *box and whiskers* plot. This diagram depicts the probability distribution of the outcomes in more compact way with respect to the probability function description. In particular, it is based on five numbers that efficiently sum up the main statistical features of the distribution:

- the *median* depicted by the red line, i.e. the value that splits exactly in half the data set;
- the *first quartile Q1*, determined by the middle value between the median and the lowest value of the data set;
- the *third quartile Q3*, determined by the middle value between the median and the highest value of the data set;
- the *minimum*, depicted by the below whisker. This does not correspond to the lowest value of the data set, but it is determined by $Q1 - 1.5 \cdot IQR$
- the *maximum*, depicted by the above whisker. This does not correspond to the highest value of the data set, but it is determined by $Q3 + 1.5 \cdot IQR$

The *interquartile range, IQR* is depicted by the blue box, and it comprehends the data between the first quartile and the third quartile. The red crosses are the *outliers*, representing the data that lies outside the whiskers range.

For example, a normal distribution will have a blue box where the median will be located exactly in the center and with equal length whiskers. Outliers will represent the 0.7% of data. [7]

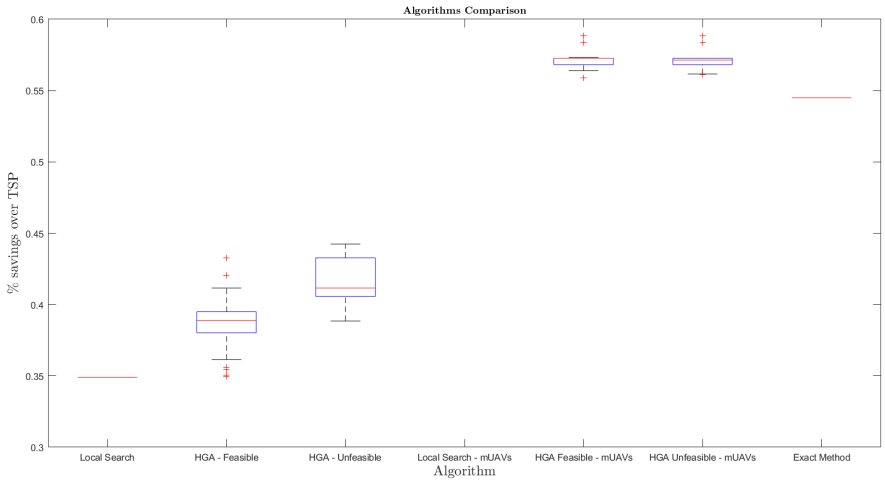


FIGURE 5.9: Results of the Monte Carlo simulation, expressed in % savings of time with respect to the TSP solution produced by the solver.

For the instance considered, where few customers need to be served and they are located in a relatively small area, all the algorithms presented show evident savings with respect to the TSP solution. In particular, among the single truck-single drone implemented algorithms, the *HGA - Infeasible Variant* has proven to be more effective, because, as pointed out in Section 4.3.3, its capabilities of exploration are enhanced by the fact that the algorithm is able to store and analyze also the solutions that would be infeasible due to drone battery life time violation. This presents an average % savings over the TSP solution of 42.18%.
Among the single truck-multiple drones approaches, no significant differences are detect among the genetic algorithms, while these are in advantage with respect to the *Exact Method* of 2.56 %. Most importantly, these methods are able halve the completion time from the TSP solution, with some solutions approaching savings as high as 60%. Furthermore, the *HGA* approaches show a good convergence, as their outcomes are tightly distributed in a narrow band around 57% of saved time from the TSP solution.
Table 5.4 synthesizes the results obtained for the Monte Carlo simulation.

	Average	σ	*Minimum*	*Maximum*
Local Search	4993	-	4993	4993
HGA - Feasible	4347	148.88	4020	4602
HGA - Infeasible	4098.05	97.21	3952	4243
Local Search - mUAVs	6700	-	6700	6700
HGA Feasible - mUAVs	3041.25	33.25	2951	3127
HGA Infeasible - mUAVs	3040.85	42.31	2916	3107
Exact Method	3223	-	3223	3223

TABLE 5.4: Statistical data for the Monte Carlo simulation made for the algorithms discussed. These data have been obtained over $n = 100$ iterated tests.

5.3 Test 3 - Algorithms Comparison - Twenty Customers Scenario

To better test the performances of the algorithms considered, a new, more challenging scenario has been examined. The coordinates have been sampled randomly, as explained at the beginning of this Chapter, and they are reported in Table 5.5.

	Coordinates	
0	41.89584681	12.67442782
1	41.89083068	12.66714406
2	41.79538085	12.66041148
3	42.00292964	12.46945934
4	41.88294013	12.4764626
5	41.83750948	12.65677788
6	41.93437279	12.35555196
7	41.86712808	12.51086024
8	41.80060169	12.56156446
9	41.80869405	12.5552257
10	41.95771275	12.34473376
11	41.93778645	12.33746581
12	41.80767318	12.5186584
13	41.91871169	12.48503364
14	41.91370674	12.55091789
15	41.90833823	12.33239965
16	41.94596145	12.4253967
17	41.84033194	12.43157198
18	41.84033194	12.43157198
19	41.83442432	12.53886858
20	41.80539241	12.37548078

TABLE 5.5: Set of coordinates generated randomly via `http://www.geomidpoint.com/random/` in a rectangular map comprehending the Rome urban area.

These coordinates have been sampled within the Rome urban area. In this way, nodes are located further apart with respect to the previous instance, thus drone time management will be more critical. Moreover, the urban area considered is expected to have traffic conditions more likely, yielding to truck travels further penalized in terms of speed of delivery.
The initial parameters have been kept unchanged, as described at the beginning of Chapter 5. As an example of graphical outcome, the map of the solution by the *HGA - Infeasible Variant* algorithm is provided along with its corresponding Gantt chart, shown in Figures 5.10 and 5.11. The average results obtained through a new Monte Carlo simulation are displayed in Table 5.6, while the outcome distributions are depicted in Figure 5.12.

FIGURE 5.10: The TSP-D solution provided by the *HGA - Infeasible Variant* for the instance created for Test 3.

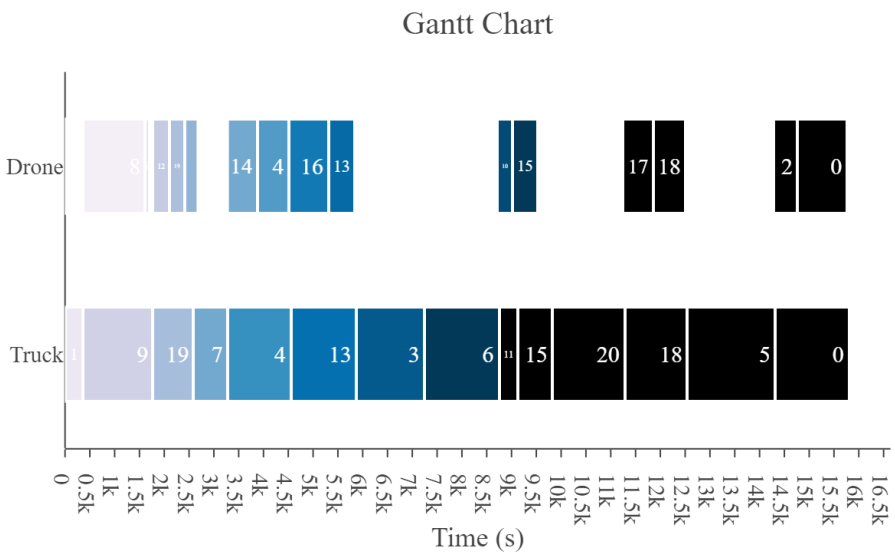


FIGURE 5.11: The Gantt chart describing timing between the UAV and the truck achieved with the solution presented in Figure 5.10.

	Completion Time			
	TSP	1 UAV	2 UAVs	*Savings over TSP (%)*
TSP solver	5 h 30 min and 0 s	-	-	-
Local Search	-	4 h 35 min and 9 s	-	16.6
HGA - Feasible	-	5 h 32 min and 57 s	-	-0.8
HGA - Infeasible	-	4 h 52 min and 6 s	-	11.5
Local Search - mUAVs	-	-	5 h 11 min and 29 s	5.6
HGA Feasible - mUAVs	-	-	3 h 40 min and 54	33.1
HGA Infeasible - mUAVs	-	-	4 h 16 min and 32 s	22.2
Exact Method	-	-	5 h 3 min and 35 s	8.0

TABLE 5.6: The results in terms of the optimized variable, the *completion time* of the delivery process, of the various algorithms that has been tested.

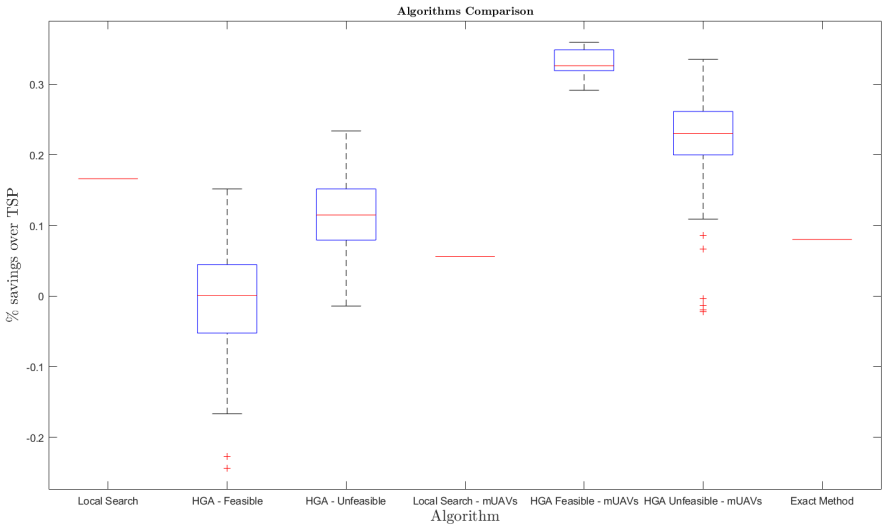


FIGURE 5.12: Results of the Monte Carlo simulation, expressed in % savings of time with respect to the TSP solution produced by the solver.

It is immediately evident that the efficiency of the algorithms is partially undermined by this new challenging scenario. In particular, the *HGA - Feasible Variant* features an average solution which is slightly worse than the TSP solution. This bad conditioning might be due to the fact that the population is initialized completely randomly, thus making the exploration of the algorithm particularly challenging, due to the higher number of combinations possible. This problem is partially overcome by the *HGA - Infeasible Variant* where the algorithm shows the potential to outrun the *Local Search Algorithm*, which performs very well in this particular instance, leading to a 16.6% improvement over the TSP solution.
The algorithm that shows the better behavior by far is the *HGA - Only Feasible Variant for mUAVs*, where it consistently reaches savings around 33.1 %. The *HGA - Infeasible Variant for mUAVs* shows instead more inconsistencies, with several outliers present below the minimum depicted by the whisker. However, the genetic algorithm stop condition could have been set later in the generations, as it will be pointed out in Section 5.4, since the number of combinations to explore are much higher with respect to the previous instances. Still $\gamma = 10000$ generations have been considered for this test in order to have consistency with respect to the simulation performed in

Section 5.2.
Statistical data of this experiment are summed up in Table 5.7.

	Average	σ	*Minimum*	*Maximum*
Local Search	16509	-	16509	16509
HGA - Feasible	19977.84	1537.07	16796	24631
HGA - Infeasible	17526.67	1060.27	15168	20078
Local Search - mUAVs	18690	-	18690	18690
HGA Feasible - mUAVs	13254.09	324.30	12679	14029
HGA Infeasible - mUAVs	15392.96	1376.37	13163	20242
Exact Method	18216	-	18216	18216

TABLE 5.7: Statistical data for the Monte Carlo simulation made for the algorithms discussed.

5.4 Test 4 - Evolution test for Genetic Algorithms

The Monte Carlo simulation iterated several times in order to get significant data related to the heuristics performances allowed to collect data on how the evolution in genetic algorithms performs generation by generation as well. In particular, the fitness values of the temporary result produced at the n generation has been stored, and it has been averaged out over the 100 simulations carried out for the previous tests. The results are depicted in Figure 5.13.

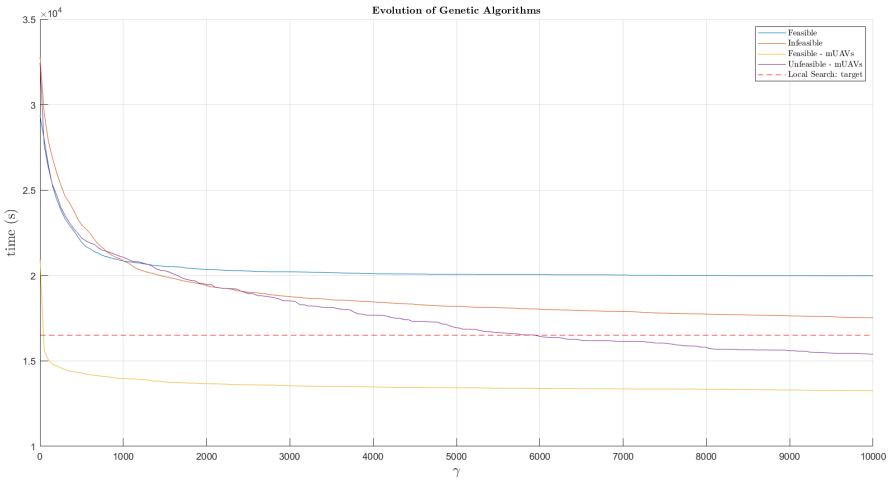


FIGURE 5.13: Evolution tendency of *HGA* algorithms over $\gamma = 10000$ iterations.

This test considers data coming from the instance described in Section 5.3. The *Local Search Algorithm*, that, as already pointed out, is performing very well for this case, has been set as a target point to be achieved by the genetic algorithms. From the plot it can be seen that the *HGA - Only Feasible Variant* features an asymptotic behavior that converges to a completion time solution which is significantly higher than the target. However, this does not hold for the *HGA - Infeasible Variant*, where still after $\gamma = 10000$ the curve has a slight downward tendency that suggests that the solution could have been improved if more generations would have been considered. This may be achieved by changing the stop conditions for the genetic algorithm that may be based on how the algorithm progressively improve or by setting

a target point upon which the algorithm will stop. In this case, the stop condition of $\gamma = 10000$ has been left unchanged from the other experiments, in order to lead to consistent results that can be compared.
The best performing algorithm, the *HGA - Only Feasible Variant for mUAVs*, has been confirmed also by this test, which shows a strong asymptotic convergence below the set target. The *HGA - Infeasible Variant for mUAVs*, even though it shows a downward tendency as its corresponding version for one drone, shows an inconsistent behavior throughout the generations, that has been highlighted in Section 5.3 as well.
A further test has been performed for the *HGA - Infeasible Variant* with one and two drones, where an increased number of generations has been considered, in order to confirm that, if these algorithms have more time to search the optimal solution, then they will converge more consistently to it. This has been suggested by the downward tendency that is still present when $\gamma = 10000$ generations are considered.
The results of this test are shown in Table 5.8.

	Completion time	**Savings over TSP (%)**
Local Search	5 h 30 min and 0 s	16.6
HGA - Infeasible	4 h 22 min and 39 s	20.4
HGA - Infeasible mUAVs	3 h 40 min and 59 s	33.0

TABLE 5.8: Results obtained setting $\gamma = 100000$ as stop condition for the genetic algorithm. As it is clear from this Table, leaving more time for the evolution of genetic algorithms leads to improved solutions.

From the Table, it is possible to observe that the convergence of the *HGA - Infeasible Variant* algorithm for a single drone towards the minimum values found in the test considered in Section 5.3 is more consistent and the same holds for the adapted algorithm operating with multiple UAVs. These results confirm that the genetic algorithms need more time to converge to optimal solutions if the scenario gets more complicated.
In the next Chapter, conclusions on the overall behavior of the considered algorithms will be drawn, as well as suggestions for possible improvements will be provided.

Chapter 6

Conclusions

The recent push towards drone employment for parcel delivery by the leading companies on the market for logistics, as well as the increasing number of researches ongoing, hints that the relatively new technology of UAVs may have a great future in this sector, either by helping the current standard vehicles to perform better, or even to substitute them. As this latter scenario, by the time of writing, seems to be immature, due to technological limitation that are still present, this Thesis has proven that the collaboration between conventional vehicles and drones is not only possible, but desirable, as efficiency of last mile delivery processes is greatly improved.
Several approaches have been investigated, starting from the heuristic approach formulated by [Murray and Chu, 2015] [13], and then, exploiting subsequent researches in the field, as the [Ha et al., 2018] [9], improvements on the initial algorithm have been implemented and tested. The provided added value from this Thesis consisted in extending these documented approaches to a multiple drones routing problem. This latter case has been explored as it will be of actual interest, since a bigger delivery fleet in turn implies larger savings, leading to economies of scale, and it is less documented in literature as well.
Overall, the methodologies analyzed performed very well with the instance composed by ten customers plus the depot that were located in relatively close proximity. In particular, a progression from the *Local Search Algorithm* to the *HGA - Infeasible Variant* algorithm towards better results has been emphasized. This has been attributed to the better exploration capacity of this last methodology over the *HGA - Only Feasible Variant*. The adaptation for multiple UAVs of the genetic algorithms has featured great performance in both cases and a strong convergence to the optimal solution. Instead, for the more complicated instance of twenty customers located further apart, a trade-off between the quality of the result and the run-time of the program has been highlighted. Indeed, the *Local Search Algorithm* showed robust performance, with a great quality of the solution to run-time ratio. Indeed, its run-time did not differ much from the ten customers instance. Instead, genetic algorithms showed a slower convergence to an optimal solution as the number of possible combinations greatly increases. Therefore, in this case, a suitable choice between a fast and robust algorithm or a slow and optimal algorithm must be taken, depending on the application.
Nonetheless, some improvements may still be applied in the algorithms implemented. In particular, the genetic algorithms stop condition may be varied according to the number of customers that must be served, so that an optimal solution is guaranteed in every scenario. The stop condition can be based on the generation index γ, as it was set in the analysis performed, or on a target fitness value that must be achieved, or finally, it can be based on the rate of improvement of the algorithm, that would stop after an arbitrary number of steps without upgrade of fitness value occurrence.
A possible enhancement of performance of the genetic algorithms, even with large

number of customers instances, may be achieved by initializing the population not randomly, but exploiting the TSP vector, so that the starting point of operations of the algorithm is already optimal in some sense.
Possibilities for future research in the field are endless, as a slight change in the initial assumptions may lead to different problem formulations and different conclusions. However, the assumptions taken in this Thesis were selected in order to lead to as much as possible faithful scenario of a real world application. Despite that, few constraints may be changed to further explore how the tandem of truck and drone for delivery behaves in different conditions. For example, as technology for autonomous vehicles advances, the launch and rendezvous locations constraints for the drone may be relaxed, leading to autonomous maneuver of the drone while the truck is traveling. Moreover, a fleet of truck may be considered in tandem with a fleet of drones. Furthermore, different cost functions may be considered depending on what is the main goal to be achieved. In many logistic applications the completion time is the fundamental parameter to be optimized, although some companies may be interested in save energy, reducing the overall emissions, save on the costs related to fuel powering the truck and electric energy feeding the drone, or safety related variables may be a reason of concern as well, as drone flight over infrastructures and even driving a truck in the road network does not come without any risk.
More realistic problem modelling may be analyzed as well. For example, the battery life may be modelled by some linear or nonlinear function of the drone speed and/or of the payload that must be carried by the drone. Similarly, the drone speed may be modelled as a three phase function, where launch and landing maneuvers are taken into account. Moreover, cruise drone speed may be affected by the payload or the energy available in the battery and these factors may be included in the model as well.
All in all, every element analyzed in this Thesis hints that a great future of drones in the logistic sector is foreseeable.

Bibliography

[1] Tom Bateman. "Drone delivery of vaccine doses speeds up COVID-19 vaccinations in remote areas of Ghana". In: *Euronews* (2021). URL: https://www.euronews.com/next/2021/06/04/drone-delivery-of-vaccine-doses-speeds-up-covid-19-vaccinations-in-remote-areas-of-ghana.

[2] Aryn Becker. "THE AMERICAN DRONES SAVING LIVES IN RWANDA". In: *Time* (2017). URL: https://time.com/rwanda-drones-zipline/.

[3] Paul Bouman, Niels Agatz, and Marie Schmidt. "Dynamic programming approaches for the traveling salesman problem with drone". eng. In: *Networks* 72.4 (2018), pp. 528–542. ISSN: 0028-3045.

[4] Wen-Chyuan Chiang et al. "Impact of drone delivery on sustainability and cost: Realizing the UAV potential through vehicle routing optimization". eng. In: *Applied energy* 242 (2019), pp. 1164–1175. ISSN: 0306-2619.

[5] Sung Hoon Chung, Bhawesh Sah, and Jinkun Lee. "Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions". eng. In: *Computers & operations research* 123 (2020), p. 105004. ISSN: 0305-0548.

[6] Júlia Cária Freitas and Puca Huachi Vaz Penna. "A variable neighborhood search for flying sidekick traveling salesman problem". eng. In: *International transactions in operational research* 27.1 (2020), pp. 267–290. ISSN: 0969-6016.

[7] Michael Galarnyk. "Understanding boxplots". In: *Towards Data Science* (2018). URL: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51.

[8] Anne Goodchild and Jordan Toy. "Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO2 emissions in the delivery service industry". eng. In: *Transportation research. Part D, Transport and environment* 61 (2018), pp. 58–67. ISSN: 1361-9209.

[9] Quang Minh Ha et al. "A hybrid genetic algorithm for the traveling salesman problem with drone". eng. In: *Journal of heuristics* 26.2 (2020), pp. 219–247. ISSN: 1381-1231.

[10] Giusy Macrina et al. "Drone-aided routing: A literature review". eng. In: *Transportation research. Part C, Emerging technologies* 120 (2020), p. 102762. ISSN: 0968-090X.

[11] Miriam McNabb. "See Amazon's New Delivery Drone Fly: Will Your Stuff Be Delivered By Drone Within Months?" In: *Dronelife* (2019). URL: https://dronelife.com/2019/06/06/see-amazons-new-delivery-drone-fly-will-your-stuff-be-delivered-by-drone-within-months/.

[12] Sergio Mourelo Ferrandez et al. "Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm". eng. In: *Journal of industrial engineering and management* 9.2 (2016), pp. 374–388. ISSN: 2013-8423.

[13] Chase C Murray and Amanda G Chu. "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery". eng. In: *Transportation research. Part C, Emerging technologies* 54 (2015), pp. 86–109. ISSN: 0968-090X.

[14] Chase C Murray and Ritwik Raj. "The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones". eng. In: *Transportation research. Part C, Emerging technologies* 110 (2020), pp. 368–398. ISSN: 0968-090X.

[15] Inc. OnFleet. "Last Mile Delivery: What it is, Trends and Tips for Success in 2020". In: *OnFleet Blog* (2020). URL: https://onfleet.com/blog/what-is-last-mile-delivery/.

[16] Andrea Ponza. "OPTIMIZATION OF DRONE-ASSISTED PARCEL DELIVERY". In: *Univeristy of Padova* (2016). URL: http://tesi.cab.unipd.it/51947/.

[17] Luigi Di Puglia Pugliese, Francesca Guerriero, and Giusy Macrina. "Using drones for parcels delivery process". eng. In: *Procedia manufacturing* 42 (2020), pp. 488–497. ISSN: 2351-9789.

[18] ReportLinker. "Drone Package Delivery Global Market Report 2021: COVID-19 Growth And Change". In: *GlobeNewsWire* (2021). URL: https://www.globenewswire.com/news-release/2021/03/08/2188710/0/en/Drone-Package-Delivery-Global-Market-Report-2021-COVID-19-Growth-And-Change.html.

[19] Mireia Roca-Riu and Monica Menendez. "Logistic deliveries with drones. State of the art of practice and research". en. In: 19th Swiss Transport Research Conference (STRC 2019); Conference Location: Ascona, Switzerland; Conference Date: May 15-17, 2019; Conference lecture on 16 May 2019. Ascona: STRC, 2019. DOI: 10.3929/ethz-b-000342823.

[20] Clifford Stein et al. *Introduction to algorithms*. eng. The MIT Press. The MIT Press, 2009. ISBN: 9780262533058.

[21] Ziye Tang, Willem-Jan van Hoeve, and Paul Shaw. "A Study on the Traveling Salesman Problem with a Drone". eng. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 557–564. ISBN: 3030192113.

[22] Sig Ueland. "8 Commercial Drone Delivery Companies". In: *Research and Markets* (2021). URL: https://www.practicalecommerce.com/8-commercial-drone-delivery-companies.

[23] Thibaut VIDAL et al. "A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems". eng. In: *Operations research* 60.3 (2012), pp. 611–624. ISSN: 0030-364X.

[24] Jeff Wilke. "Amazon moves closer to its goal of a drone delivery solution that scales to meet the needs of customers." In: *Amazon* (2019). URL: https://www.aboutamazon.com/news/transportation/a-drone-program-taking-flight.

[25] Laura Wood. "Worldwide Drone Delivery Industry to 2030 - Increasing Need for Contactless Deliveries for Safety Purposes is Driving Growth". In: *GlobeNewsWire* (2021). URL: https://www.globenewswire.com/en/news-release/2021/04/06/2204790/28124/en/Worldwide-Drone-Delivery-Industry-to-2030-Increasing-Need-for-Contactless-Deliveries-for-Safety-Purposes-is-Driving-Growth.html.