UNIVERSITÀ
DEGLI STUDI
FIRENZE

UNIVERSITÀ
DEGLI STUDI
PERUGIA

Istituto Nazionale
di Alta Matematica

Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

**DOTTORATO DI RICERCA**

**IN MATEMATICA, INFORMATICA, STATISTICA**

CURRICULUM IN INFORMATICA

CICLO XXXVI

**Sede amministrativa Università degli Studi di Firenze**

Coordinatore Prof. Matteo Focardi

# Optimizing Drone-Based Applications for Delivery and Smart Agriculture

Settore Scientifico Disciplinare INF/01


**Dottorando**:

Lorenzo Palazzetti

**Tutore:**

Prof. Maria Cristina Pinotti

**Coordinatore:**

Prof. Matteo Focardi


Anni 2020/2023

# Contents

# List of Figures

# List of Tables

# List of Algorithms

*I dedicate this dissertation to my Granpa.*
*I miss you more than words can say.*
*Thank you for believing in my dreams.*

# Abstract

The intersection of Unmanned Aerial Vehicles (UAVs), commonly known as drones, with various human labor-intensive activities such as last-mile delivery, agriculture, and surveillance has emerged as a compelling frontier of research. This dissertation explores the transformative impact of drones in two key domains: last-mile delivery and agriculture.

In the context of last-mile delivery, drones are rapidly reshaping logistics practices, offering opportunities to overcome geographical barriers, reduce delivery times, and minimize carbon footprints. The cost-effectiveness and adaptability of drones make them accessible to businesses and communities. However, this transition to drone-based delivery systems is not without its challenges. The first part of the dissertation focuses on last-mile delivery, exploring innovative solutions to optimize operational efficiency and enhance sustainability. Chapter 1 investigates the impact of wind dynamics on drone-based delivery systems defining a tunable model for representing drone energy consumption according to wind conditions. Then, an algorithm to plan minimum-energy trajectories, and one to analyze mission feasibility for drones in windy conditions are proposed. Chapter 2 explores the cooperation between a truck and a fleet of drones in the context of last-mile package delivery by introducing the Scheduling Conflictual Deliveries Problem (SCDP). The SCDP models each delivery as an interval, and each drone as channel of limited capacity, i.e., energy, and aims to maximize the collected reward associated to each delivery, i.e., the delivery priority. Finally, optimal and approximation algorithms are provided presented with a thorough performance evaluation on synthetic data.

The second part of the dissertation delves into agriculture, specifically crop monitoring, using drones to collect data efficiently from specific locations. The possibility to visualize with constant update the crop health state helps the farmer in the decision process allowing to take actions and measurements promptly towards the root of the problem. By discretizing the observation positions inside a crop, the monitoring task can be reduced to an Orienteering Problem (OP) where a reward function shapes the relevance of visiting certain positions, and a cost function represents the energy consumption due to drone operations. Indeed, solving the OP requires visiting the subset of locations associated with the maximum relevance without exceeding the energy bud-

get for each drone. Under this framework Chapter 3 presents a solution for the data collection problem in a special orchard type, i.e., modern orchards protected by nets, providing a novel data structure to model it, and introducing the Single-drone Orienteering Aisle-graph Problem (SOAP), a specialized version of the OP. In SOAP, the objective is visiting the most relevant observation points in the orchard for collecting images without exceeding the battery capacity of the drone. An optimal, two approximation algorithms, and two fast heuristics have been developed to solve it. Chapter 4 continues the same research by modeling the data collection problem as the task of offloading the data collected by a set of sensors with drones. In detail, a wireless sensor network (WSN) strategically deployed within monitoring areas is in charge of collecting useful information on several physics phenomena. Since the deployment area can be very large, we rely on a set of homogeneous drones to selectively collect the data from the sensors under two main limitations. First of all, a drone during its data harvesting mission is subjected to the battery capacity, i.e., it has a maximum flight range. Secondly, the available memory storage is limited, so it can store only a limited amount of data during its mission. As a result, we obtain a novel optimization problem, the Multiple-drone Data-collection Maximization Problem (MDMP) which represents a further generalization of the OP, where we optimize also the drone storage capacity. The problem is proved to be NP-hard, and for it are proposed an optimal ILP solution, an approximation, and two fast heuristics.

The third part of the dissertation focuses on smart agriculture, particularly pest detection in orchards. The invasive Halyomorpha Halys (in short HH), commonly known as Brown Marmorated Stink Bug, poses a significant threat to agriculture. The dissertation details the development of an Integrated Pest Monitoring (IPM) system, combining drones and machine learning algorithms for pest detection, with a specific emphasis on this invasive bug. Chapter 5 concentrates on improving pest detection using machine learning models trained on field-captured images. To the best of our knowledge, we explore for the first time the possibility to detect this invasive bug employing a computer vision algorithms trained on images taken inside orchard aisles. The chapter covers every phase of the training process, ending with wide performance analysis based on several factors, including variations in blurriness levels. Finally, we devise a cloud service architecture for extending the accessibility of the algorithms devised. Chapter 6 focuses again on the creation of a detection algorithms for scouting the HH, posing a special attention on the automatization of the entire process, i.e., mission planning, data acquisition, and bug detection. Therefore, we first define a drone-based protocol which allows to collect images from the top of the orchard, at an altitude of 10 meters, pushing on strict camera settings, and photograph techniques. To the best of our knowledge, relying on this protocol we create the first datasets for scouting HH completely

based on drone-captured images. Then, employing a tailored training algorithm based on image slicing and sub-portion attention we train several models exploiting the YOLOv5 algorithm. Finally, we analyze performance of the networks and models predictiveness using well-known and custom metrics. Throughout the dissertation, the fusion of technology and agriculture is highlighted, paving the way for a more sustainable and efficient future in farming.

The research contributions discussed in each chapter have been published in various conferences and journals, showcasing the practical relevance and impact of the work. Overall, this dissertation addresses critical challenges and provides innovative solutions for the integration of drones into last-mile delivery, agriculture, and pest monitoring, contributing to the ongoing evolution of drone-based systems in these domains.

# Introduction

In an era defined by rapid technological advancements and innovations, the intersection of unmanned aerial vehicles (UAVs), commonly known as drones, and the increasing difficulty for finding workforce in a variety of everyday life activities, e.g., agriculture, delivery, and surveillance, has emerged as a compelling frontier of research exploration. Indeed, The widespread adoption of robots and artificial intelligence (AI) is revolutionizing multiple industries and will likely reshape many aspects of our lives. Automation, industrial robots, and the forthcoming AI revolution have already changed the nature of work and production processes. Some studies have highlighted the positive effects on productivity, growth, and long-run outcomes [2]. This reallocation of tasks may directly affect work-related health risks as well as workers' perception of job safety as robots are often used in tasks that are physically intense and may involve higher risks. For example, the use of drones in last-mile delivery is rapidly transforming the conventional logistics practices, and redefining the way we perceive and interact with this sector. Indeed, the introduction of this technologies is shifting the human role from active part in the field where everything depends on the employees, to a more passive position where the activities are demanded to the drones, and the human presence servers mainly as supervision, ensuring that every task goes smooth. Moreover, with an ever-growing demand of efficient, timely, and sustainable delivery solutions, drones offer an unprecedented opportunity to reshape the logistics landscape. They pose the premise to overcome geographical barriers, reduce delivery times, and minimize carbon footprints. Furthermore, the cost-effectiveness and adaptability of drones, more and more cheap drone models are available off-the-shelves, in this context can unlock new levels of convenience and accessibility for customers, businesses, and communities alike.

Simultaneously, the application of drones in agriculture opens new horizons for optimizing farming practices and increasing agricultural productivity. Through real-time monitoring, data collection, and precision agriculture techniques, drones empower farmers to make informed decisions, optimize resources allocation, and last but not least drastically reducing the time spent for tedious tasks that require a constant human presence, e.g., field sowing, crop harvesting. Today more than ever, in a world where the demand for food is constantly increasing, the efficiency

and the quality of the agriculture production is a crucial challenge to pose new food production standards, and reducing the escalation of demands.

Yet, as with any pioneering technology, the road to realizing these beneficial innovations is not without its share challenges. Ethical, regulatory, safety, and technical hurdles loom large on the path to the seamless integration of drones into last-mile delivery and agriculture. This dissertation does not shy away from addressing these challenges head-on, providing a comprehensive examination of the complexities that must be navigated to harness the full potential of drones in these two domains. Furthermore, it discusses solutions and their relative efficiency for overcoming the aforementioned challenges acting as flywheel for further improvements.

The dissertation is organized into three parts, covering the introduction of efficient and effective solutions for improving the usage of drones in last-mile delivery, data collection for crop monitoring, and insect detection for a pest management system. The Part I of my dissertation embarks on a journey through two interconnected chapters, each offering a innovative perspective on the evolving world of drone-based delivery systems. In these two chapters, we explore solutions to optimize operational efficiency in last-mile deliveries by investigating the energy impact of wind on drones trajectory planning, and proposing for an optimal algorithm for mission planning. Hence, we demonstrate that the wind, seen by many as a limitation, can be converted to an invaluable resource for flight time extension, and as consequence, an effortlessly way to enhance the sustainability of last-mile package delivery. Under this framework, we navigate strictly related critical challenges, such as the delivery feasibility analysis in cooperative scheme with a truck and a drone. In addition, we provide rigorous algorithmic solutions towards the delivery scheduling optimization using in a truck and drone delivery system.

More in detail, in Chapter 1, we delve into a fundamental yet often overlooked aspect of drone-based delivery systems: the impact of wind dynamics. As drones become increasingly integral to last-mile delivery, understanding and adapting to the influence of wind patterns become essential for optimizing their flight endurance. Our research takes a pioneering step in this direction, where, for the very first time, we adapt drone trajectories to harness the power of the wind.

We consider a scenario where a truck and drone work in tandem, with the drone actively responding to wind conditions to adopt the most energy-efficient trajectory between the truck's path and the delivery location. This novel approach gives rise to the Minimum-energy Drone-trajectory Problem (MDP), a challenging optimization problem. MDP aims to plan minimum-energy trajectories for drones, particularly when wind affects the delivery area. Within this chapter, we introduce two distinct algorithms that provide optimal solutions to MDP under varying truck routes. Additionally, we delve into the feasibility of employing drones with limited battery capacity for

deliveries. To validate our proposed algorithms and model, we conduct numerical comparisons using synthetic and real data, along with simulations in the BlueSky simulator.

The Chapter 2 of this dissertation delves into the intricate symbiosis between a truck and multiple drones in the context of last-mile package delivery. Here, we introduce the Scheduling Conflictual Deliveries Problem (SCDP). In this scenario, a truck plays a pivotal role in transporting a fleet of drones, each tasked with package deliveries. The truck follows a predefined route, while each delivery task is characterized by drone energy consumption, priority, and temporal constraints tied to the truck's route.

Our primary objective in SCDP is to optimize the scheduling of drones to maximize overall reward, all the while adhering to strict battery capacity constraints and ensuring non-intersecting delivery intervals. Recognizing the computational complexity of this problem, we establish its NP-hard nature. To tackle this challenge, we present an Integer Linear Programming (ILP) formulation and devise a pseudo-polynomial time optimal algorithm for the single drone scenario. Furthermore, we offer approximation algorithms suitable for both single and multiple drone scenarios. To assess the practicality and performance of our proposed algorithms, we undertake a comprehensive comparative analysis using diverse synthetic datasets.

These two chapters collectively contribute to the ongoing evolution of drone-based last-mile delivery systems. They introduce innovative solutions, address critical challenges, and pave the way for more sustainable, efficient, and adaptive delivery systems in the modern era.

The accomplishments discussed in Chapter 1, pertaining to wind modeling for drone delivery and mission feasibility, were initially presented in two conference papers [3,4] and subsequently explored in more depth in the corresponding journal publication [5]. Additionally, the code that was developed can be found in the following GitHub repository (https://github.com/TheAnswer96/T-ITS23_bluesky-plugin). As for the content covered in Chapter 2, which deals with the scheduling problem within the context of last-mile deliveries, it was initially examined in two conference papers [6] and later enhanced in the subsequent journal paper [7].

The Part II of the dissertation spans over the problem of collecting data from specific locations of a crop with different degrees of drone's mobility. As the data demanding requires to be increasingly heavy to have a better insights and inferring accurately the crop health with extreme precision, past solutions that rely on wireless Sensor Network (WSN) or even worse on human has started to suffer from the burdening of the huge volume of data. The introduction of drones for crop monitoring defines a turning point concerning performance, posing a new efficiency standard with respect to health plant operators inspection, as well as employing them as mobile sink, i.e., a data carrier, represents the definitive crux for optimizing and extending WSN life-time. In

principle, the problem of collecting data can be modeled as well-known optimization problem, the *Orienteering problem* (OP) [8]. The OP is a routing problem in which the goal is to determine a subset of nodes to visit, and in which order, so that the total collected score is maximized and a given time budget is not exceeded. In the following two chapters, we consider two different data collection problem, and by demonstrating their connection to special version of the OP, we depict for them efficient solutions, proving by accurate simulation their valuable advancement.

In detail, in Chapter 3 we face the challenge of using drones for collecting relevant data within special orchards, particularly common in Italy, protected by nets. To navigate this challenge, we investigate the chance of modeling the orchard as an aisle-graph, by defining a novel data structure, i.e., the *3D single-access aisle-graph*. Then, we formalize the problem of visiting the most relevant position of the orchard under the drone energy constraint as a novel variant of the orienteering problem, the *Single-drone Orienteering Aisle-graph Problem* (SOAP). We demonstrate that SOAP can be optimally solved in polynomial time, and more efficient solutions have been proposed in order to deal with large problem instances. Finally, through extensive evaluation that exploits synthetic and real-world data, we showcase the effectiveness and efficiency of our solutions.

Chapter 4 delves into the realm of Internet of Thighs (IoT) sensors strategically deployed within areas to be monitored. Here, we explore the possibility of employing drones in collecting data from these sensors, under real-world constraints, e.g., drone energy and storage limitations. We introduce the *Multiple-drone Data-collection Maximization Problem* (MDMP), where the objective is to plan the mission for a set of drones that maximizes the *relevance* of the data collected while adhering to the energy and storage constraints. Then, an Integer Linear Programming algorithm for optimally solving the MDMP is provided, as well as further suboptimal algorithms for the single and the multiple drone scenario are presented. As already done for the previous chapter, we validate the theoretical performance of the proposed algorithms through accurate simulations.

The accomplishments discussed in Chapter 3, concerning the algorithm designed for optimizing the pest monitoring in a modern orchard enveloped by nets, were initially presented in two conference papers [9,10] and subsequently explored in more depth in a scientific contribution currently submitted for possible publication in IEEE TMC Additionally, the code that was developed can be found in the following GitHub repository (`https://github.com/TheAnswer96/TMC`). The content covered in Chapter 4, which deals with the drone path optimization towards collecting the most relevant data harvested from a Wireless Sensor Network (WSN), it was initially examined in the single drone scenario in [11] and later generalized in the subsequent multiple

drone scenario in the journal paper [12]. As regards the code, it can be found in the following GitHub repository (https://github.com/TheAnswer96/JCSS).

Finally, the Part III of this dissertation delves into the prominent field of smart agriculture, with a special focus on pest monitoring. Pest monitoring is becoming increasingly important, and is getting through a revolution, and a modernization due to the new hot technologies, e.g., drones, neural networks. Agriculture, tracing its evolutionary path from the mechanical revolution onwards, has witnessed a steady infusion of technological advancements, culminating in ever-more powerful machinery and enhanced mechanical expertise. Nevertheless, it has lagged behind other sectors of the economy in harnessing state-of-art technologies and automation as pointed out in [13].

In the pages that follow, this dissertation concentrates its focus on the careful selection and deployment of innovative technologies tailored to address several pressing concerns that arise in the context of crop monitoring. Specifically, our efforts center on the development of a Integrated Pest Monitoring (IPM) specialized for a particularly invasive and harmful insect, the Halyomorpha Halys (HH), colloquially known as "brown marmorated stink bug" (BMSB).

This invasive species, native to regions in East Asia, including China and Japan [14], poses a significant threat to agriculture due to its polyphagous nature, feeding voraciously on various host plants, especially fruit-bearing trees, e.g., pear, hazelnut trees.

The emergence of the HH as a global pest is attributed to factors such as global trade, and side effects of climate change [15, 16]. Italy, renowned for its fertile orchards, encountered the destructive potential of HH starting by 2012, marking the official entry of this pest into European territory. The ensuing years have witnessed substantial losses in fruit production and in trading revenue, with 2019 estimates reaching a staggering €588 million only in northern Italy.

Efforts to monitor and mitigate HH infestations have relied on methods ranging from traps baited with aggregation pheromones, which have proven somewhat unreliable and sometimes have inadvertently increased damage, to time-consuming active monitoring techniques, such as the well-known estimation technique of *frappage* where a plant health operator repeatedly beat on trees' trunk, estimating the HH presence by counting how many specimens fall down. Attempts to combat the HH outbreaks have, regrettably, led to a surge in the use of a broad-spectrum pesticides, disrupting existing IPM programs, escalating producer costs, and inflicting relevant harm not only for the consumers but also for the environment.

This part of my dissertation presents the line of research pursued as regard the Horizon 2020 HALY.ID project [17], granted with the objective of automating the monitoring activities of growers and plant health operators. The HALY.ID project aims to mimic the monitoring process just

described, exploiting innovative technologies. Under the umbrella of the project has been deployed architecture which is needed to collect the required data for training the machine learning (ML) algorithms, and to autonomously monitor the field. Furthermore, to monitor the abundance of HH in orchards, various hardware and software solutions are proposed and implemented by multiple research units involved in the project. One notable solution is the use of a Unmanned Aerial Vehicle (UAV) equipped with a high-resolution camera to capture images of the insects on trees, which are then compiled into a dataset utilized then by ML algorithms to identify the presence of HH. Also, an Internet of Things (IoT) sticky trap, and a network of IoT-based microclimate stations are established to continuously monitor pest populations.

Under this premise, we will navigate through two pivotal chapters, each shedding light on a unique facet of our overarching goal – the automation and the enhancement of the monitoring of Halyomorpha halys.

In detail, Chapter 5 focuses on combining drones and Deep Learning algorithms for pest detection, with a specific emphasis on the detecting the Halyomorpha Halys. While the integration of autonomous drones and advanced vision chips into integrated pest management holds great promise, it also brings tough challenges. In this chapter, we detail our efforts to enhance the performance of a detection algorithm relying only a dataset comprised by only on-field captured images. We conduct a rigorous data cleaning process based on two crucial quality parameters, blurriness and brightness. Subsequently, we train and evaluate various ML models within the YOLO framework, employing diverse metrics for performance comparison. Finally, we conduct a thorough experiment designed to shed light on the role of blurriness in insect detection performance.

Chapter 6 devotes attention on the challenging task of bug detection, i.e., Halyomorpha Halys, exploiting a drone-based automated protocol. The chapter states a new standard for data harvesting, which tackles the main limitation encountered in the previous chapter, namely flying within orchard's aisles. Indeed, the maneuvers of bulky drone in tight spaces represent the main culprit of drone damages, and very often cause flight deadlock when the collision avoidance system is engaged. For this reason, according to several optics concepts a data harvesting procedure from the top of the orchard is defined. This procedure allows the creation of the first dataset for Halyomorpha Halys detection with high-resolution images taken from the top of an orchard by a drone. Although the attention put to correctly adhere to all the specification of the orchard, a cleaning procedure based on the most relevant parameters, i.e., blurriness and brightness, was required. Then, since the bug covers a very limited portion of the images a tailored training process based on image slicing is performed in order to put attention on sub-portion of the original

4K image, ensuring any information loss. Finally, we evaluate the ML models trained within the YOLO framework, employing diverse metrics for performance comparison.

As you navigate through these chapters, you witness the fusion of technology and agriculture, paving the way for a more sustainable and efficient future in farming.

The achievements described in Chapter 5 and Chapter 6, spanning on the development and the deployment of an insect detection algorithm trained only in-field images, were firstly published exploiting more basic strategies in [1, 18, 19], and then, improved by employing a tailored pre-processing, and more sophisticated techniques in the journal paper [20].

# Part I

# Drone Trajectories Wind-aware and Drone-based Deliveries

# Chapter 1

# How the Wind Can Be Leveraged for Saving Energy in a Truck-Drone Delivery System

## 1.1 Introduction

Recently, drones or Unmanned Aerial Vehicles (UAVs) have been widely investigated in civil applications, such as agriculture [21], environmental protection [22], task offloading [23], localization [24, 25], last-mile package delivery [7, 26], and so on. In fact, drone-based solutions for the delivery of small packages have been announced by big companies in e-commerce. Initially, it did not seem safe or practical to have robotic objects flying in the sky, but now there are several reasons to believe that package delivery by drones may be coming soon. In fact, new regulation laws have been just released in 2021 by the Federal Aviation Administration (FAA), which allow operators of small drones to fly over people and at night under certain conditions [27].

Drones are considered to be much faster than traditional trucks in some circumstances and can also lead to a reduction in pollution and greenhouse gas emissions. For example, drones are not delayed by rush hours in congested areas and can significantly shorten the paths to reach customers by flying over rivers, green parks, and sea bays. So, it is highly expected that pretty soon, if not already, companies can further extend their business by relying on drones that cross the last mile to their customers. However, drone-based delivery systems still have to face exciting challenges. First, due to the small size of the battery, drones (i.e., copters) have limited flight autonomy, and such a limitation is more accentuated by their carried payload. Also, due to payload constraints, drones cannot fulfill more than a single delivery at a time, so they must go back to a depot for grabbing other packages for other customers. Finally, weather conditions, in particular *wind*, must be considered during the drones' deliveries.

In this chapter, we concentrate on the wind challenge. Surprisingly, in the literature, the wind has been only considered as a limiting factor for drones, to the best of our knowledge. All the manufacturers set the maximum allowed wind speed that can be safely tolerated by their drones.

The wind as a factor that influences the aircraft's path has been previously considered in [28]. The authors share our interest in finding the "global optimal path" in the presence of wind. However, their aim is to optimize the duration of the flight, specifically, they focus on determining the aircraft's minimum time path while also avoiding obstacles. The aircraft's energy is not taken into account. Moreover, the kind of the aircraft and its routes are not comparable with ours. Namely, the geographical scale is completely different. The authors in [28] analyze the wind effect on commercial planes, powered by fuel, whose routes' lengths are much longer than ours (i.e., $\geqslant$ 800km).

In this chapter we provide a proof-of-concept of how wind can help drones during deliveries. We consider the following scenario, which is realistic in the current state of technology. We assume drones engaged in last-mile deliveries with lightweight payloads. In a delivery, a drone can afford to carry 1–6kg of payload in a range of 8–12km [29]. To reduce the length of the round trips to/from the depot, inspired by big companies' drone delivery programs, we assume that a drone is assisted by a truck. The drone picks up the packages from the truck and flies towards the customer, and then back to the truck, while the latter continues its journey near a series of delivery points [30–32]. In the literature, several scenarios have been addressed using truck-drone tandem delivery systems, but at the best of our knowledge, no one considers adapting drones' trajectories to wind conditions.

Motivated by the observation that when the drone operates under "tailwind" conditions (i.e., when the wind blows in the direction of the drone's movement) it consumes less energy, we explore the possibility that the drone selects the detaching point from the truck to reach the customer following, as much as it can, the "tailwind" trajectory. We do that even at the cost of increasing the length of the trajectory, as long as we do not lose energy. The same approach is followed to return back to the truck (see Figure 1.1).

Under this premise, we assume that the drone's route consists of a *polygonal chain* that starts from the *take-off point* $T$ on the truck's road and returns to the *landing point* $L$ passing through the delivery point $P$ (see Figure 1.1). Notice that, in general, the take-off and landing points do not coincide. We only assume that the take-off point precedes the landing point on the truck's route. To give the drone the freedom to arbitrarily select the take-off and landing points, we assume a rural or suburban environment, like a residential area neighborhood, where already at low altitudes, the drone can freely move in any direction having no significant obstacles on its

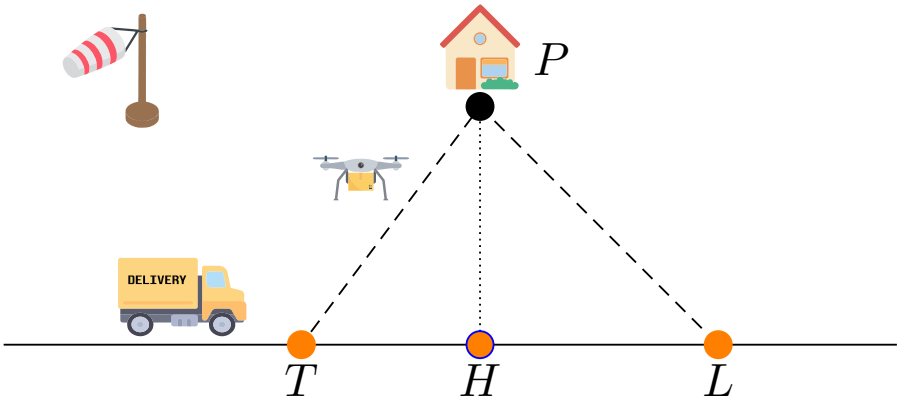



Figure 1.1: Abstraction of a delivery performed by the truck-drone tandem in a supply chain scenario. The truck moves carrying the drone; $P$ is the delivery.

way. We also assume a very mild truck-drone synchronization, and so, the first vehicle that arrives at the landing point, waits for the other.

Our goal is twofold: i) to determine the take-off and landing points on the truck's route that minimize the energy consumed by the drone for the delivery, and ii) to study the delivery feasibility because the drones are energy-constrained. The results obtained in this chapter can be summarized as follows:

- We define the Minimum-energy Drone-trajectory Problem (MDP) whose goal is to find the minimum-energy trajectories when the wind influences the delivery area;
- We devise two algorithms for MDP, i.e., SINGLE and MULTI which assume that the truck's path is a straight line or a polygon, respectively;
- We analyze the feasibility in terms of energy of sending drones for deliveries in a windy area, by presenting the FEASIBILITY algorithm;
- We numerically compare our algorithms on synthetic and real data sets. Moreover, we evaluate the energy spent by our solution simulating the drone's flight in BlueSky [33].

The rest of the chapter is organized as follows. Section 1.2 surveys the related work. Section 1.3 introduces the wind and energy models. Section 1.4 defines the MDP and details the math that supports the SINGLE and the FEASIBILITY solutions. Section 1.5 gives the pseudo-code and the flowcharts of the SINGLE, the MULTI, and the FEASIBILITY algorithms. Section 1.6 numerically evaluates our algorithms through synthetic and real winds, and also simulates the drone's flight by using BlueSky. Finally, Section 1.7 offers conclusions.

## 1.2 Related Work

In this section, we review works about truck-drone delivery systems with an emphasis to those influenced by the wind.

### 1.2.1 Truck-Drone Tandem Delivery Systems

Recently, the problem of delivering goods with drones has been approached by several papers [31, 34]. The vast majority of them assume that the drone has a limited battery and has to return back to the warehouse after every delivery due to the payload and energy budget constraints. Many solutions consider drones working in tandem with a truck, and there are many ways the two means of transportation can collaborate. Some solutions divide the deliveries between the two, while others make them collaborate in every single mission [35–37].

The authors in [35] consider symbiotic cooperation between a truck and a drone to accomplish a set of deliveries. Precisely, the goal is to minimize the makespan to serve all the customers by either the truck or the drone. This problem is presented as a variant of the TSP known as the flying sidekicks traveling salesman problem. The authors propose a hybrid heuristic divided into three steps. Initially, the algorithm creates a TSP solution where the truck serves all the customers. Next, the algorithm greedily assigns deliveries to the drone considering the quantity of time saved with respect to the truck without exceeding the battery budget. Each delivery is served with a polygonal chain that starts from and returns to a point of the truck's route passing through the customer. Finally, the algorithm tries to improve the solution by visiting the neighborhood of each customer, i.e., by evaluating a different assignment for each customer. Our work is completely different. Although like [35] we use a polygonal chain to serve a delivery customer, we select the polygonal chain by minimizing the energy for every single delivery and not to minimize the total makespan of all the customers. Moreover, in our solution, any position on the truck's road is a candidate for the take-off/landing, whereas in [35] only a subset of positions, which correspond to points on the truck's route, are candidates.

A similar problem is proposed in [37] where, given a fixed sequence of stops that constitute a truck's route and a set of customers, the goal is to find a scheduling for drones (i.e., a set of trips defined by a drone's take-off, customer service, and landing) so that all the customers are served and the makespan is minimized. The authors provide a mixed-integer programming formulation to solve the single and multiple drone scenarios. However, unlike our approach, it is not possible to change neither the truck's route nor the drone's trajectories for a customer in order to increase time savings.

Notice that, all these works do not consider the impact of winds on energy consumption since they are focused only on the time required to perform a delivery.

### 1.2.2 Drone Delivery Systems Influenced by the Wind

The impact of the wind on defining long-distance trajectories of aircraft has been considered in [28]. Instead, the impact of winds on small and battery powered drones' trajectory planning has not received much attention so far.

In [3, 38, 39], the wind is introduced in the model. Precisely, the drone trajectories are represented by paths in a weighted graph whose edge weights model the energy consumed by the drone, which depends on the carried payload and the wind speed. In [3,39] authors focus on finding which is the percentage of deliveries that can be accomplished with a given energy budget knowing the wind conditions on-the-fly. Authors of [40] extend the above solution to a multi-depot multi-drone delivery system. Whereas in [3,39], the drone moves along predefined routes represented by the edges of a graph given in input to the problem, in the problem proposed in this chapter we must find the drone's trajectory, which is the output of the problem, by leveraging the impact of the wind.

A multi-drone delivery system dealing with wind, obstacle avoidance, and maximum energy budget, is also proposed in [41]. A subset of feasible deliveries is initially computed offline, and a depth-first search strategy is performed then to get the final mission plan. Their extended work [42] considers the same setting with many clusters, in which the problem is seen as a constraint satisfaction problem. In [43], a delivery scenario where the wind affects multiple drones is considered. Differently from our approach, each drone can perform multiple deliveries on the same mission. The focus is on optimizing the global mission plan. The wind influences the energy consumption, but the drone passively suffers the wind, i.e., it does not adapt the route to the wind, as we do.

Lastly, authors in [44] exploit the service paradigm to abstract drone's capabilities into a drone service. The main novelty is the creation of a test-bed that consists of an indoor replica of a city. A drone flies over it by using different polygonal trajectories and payloads. The wind conditions are simulated by a fan. The energy consumption is inferred from the collected data of 72 indoor flights. The authors focus on the study of how the shape (triangular, rectangular, hovering) of the drone's path acts on the energy consumption.

In conclusion, the content described in this chapter is different from the surveyed papers in many aspects. The most important difference is that in our study the battery powered drone actively reacts to the wind and searches for the minimum energy trajectory, i.e., the trajectory as

closer to the tailwind as possible given the truck's movement. Computed the optimum trajectory, we compare how much energy is saved using the found trajectory with respect to the energy spent traversing the shortest trajectory, which is the best solution in absence of wind. Hence, the main research gap we are filling is the formalization of a model able to handle and understand how the wind affects the drone's flight. Furthermore, to the best of our knowledge, this is also the first time that wind has been taken into account while creating an algorithm to determine the optimal drone's route.

## 1.3 Background

In the background of our solution there is the *wind triangle* concept. In air navigation, the wind triangle (see Figure 1.2) is a graphical representation of the relationship between the drone's motion and the wind. The *ground vector* represents the motion of the drone over the ground, and

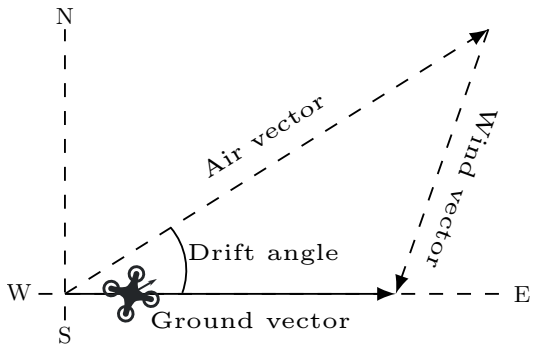



Figure 1.2: The wind triangle.

it is the resultant of adding the *air vector*, i.e., the motion of the drone through the air mass, and the *wind vector* [45]. Specifically, when there is a wind and the drone wants to follow a specific ground track, the air vector indicates the true heading and drone's speed. The difference in angle between heading and ground track is known as the *drift angle*. So, if the drone wants to follow the desired track, it must correct its heading of an angle depending on the wind direction. Each vector $\vec{z} = (z_d, z_s)$ in the wind triangle is characterized by a direction $z_d$ and a speed $z_s$.

We first observe that in presence of wind, varying the ground direction also varies the air vector (see Figure 1.2). Since we aim to find the track that minimizes the energy spent by the drone (i.e., we try many possible ground directions) and since the energy spent to fly depends on the air vector (as we will see in Eq. (1.1)), our solution relies on the aforementioned wind triangle construction. The second observation is that by varying the ground track, the energy changes due to the variations in both the length of the trajectory to be traversed, and the air speed. In this

chapter, we will show that sometimes, in presence of wind, it is convenient to leave the shorter drone's trajectory for a longer one that consumes less energy.

In the rest of this section, we explain how we derive the unitary energy for the ground trajectories.

### 1.3.1 The Relative Wind and Energy Model

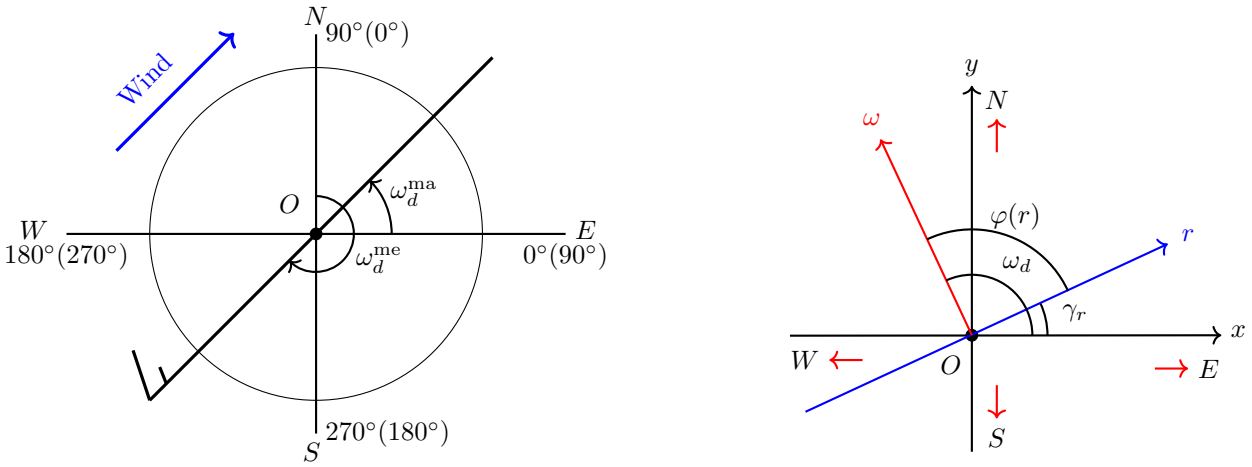
The wind, characterized by speed and direction, is an important variable for evaluating the drone's energy consumption. Usually, the weather stations record the *meteorologic direction* of the wind $\omega_d^{\text{me}}$, i.e., the direction from which the wind originates, assuming the North as the $0^\circ$ direction. For instance, if the wind blows from the North to the South, a weather station records a direction of $0^\circ$, while if it blows from the East to the West, the weather station refers to $90^\circ$. From now on, to ease the notation, we indicate the angles without the degree symbol ($\circ$). The meteorologic direction of the wind conventionally grows clockwise in a Cartesian coordinate system $xOy$ whose $x$-axis is the North direction and the wind direction is given as the direction the wind is from. To conform the wind to the classical Cartesian coordinate system, we convert the meteorologic direction of the wind into the *mathematical direction* of the wind $\omega_d^{\text{ma}}$. First, $\omega_d^{\text{ma}}$ defines the winds blowing to the origin $O$ of the Cartesian coordinate system and not from it. The mathematical wind directions grow counterclockwise from the usual $x$-axis which is equal to the East direction. Hence, the conversion rule from the meteorological direction of the wind to the mathematical one is explained in [46], and it is as follows:

$$\omega_d^{\text{ma}} = \Big| \underbrace{-\omega_d^{\text{me}} + 360}_{\text{clockwise}} \underbrace{+90}_{\text{phase}} \underbrace{-180}_{O} \Big|_{360} = |-\omega_d^{\text{me}} + 270|_{360}.$$

Figure 1.3a shows the direction of the meteorological wind $\omega_d^{\text{me}} = 225$ that is directed towards $O$ and the corresponding mathematical direction $\omega_d^{\text{ma}} = 45$ that has $O$ as the source. In this chapter, from now on, we always refer to the mathematical direction of the wind.

Let us now define the *global wind* $\omega$ as the (mathematical) wind that blows in the delivery area. The global wind vector $\omega = (\omega_d, \omega_s)$ has *direction* $\omega_d$ and *speed* $\omega_s$. Notice that, from now on, we assume to leverage the built-in sensing system on the drone for estimating the global wind vector prior to the take-off. The *relative wind on* $r$ $\varphi(\gamma_r)$ has direction $|\omega_d - \gamma_r|_{360}$ which is the difference mod 360 between the direction of the global wind and the direction of the drone's trajectory (see Figure 1.3b).

The energy for flying a unit of distance (e.g., 1m) on a line $r$ mostly depends on the air speed of the drone, which in turn depends on the global wind $\omega = (\omega_d, \omega_s)$ and on the ground vector




(a) (b)

Figure 1.3: Axes are labeled with the cardinal, mathematical, and meteorologic (in brackets) directions. Note the *phase* (equals to 90) which represents the offset North-East among the two ways for representing the wind (a); Relative wind $\varphi(\vec{r}) = 90$ on $\vec{r}$ when $\gamma_r = 25$ and $\omega_d = 115$ (b).

$\vec{r} = (\gamma_r, v_d)$, as seen in Figure 1.2. Precisely, the *air speed* $a_s$ can be calculated as [29]:

$$a_s = \sqrt{s_N^2 + s_E^2}, \tag{1.1}$$

where $s_N = v_d - \omega_s \cos(\varphi(\gamma_r))$ and $s_E = -\omega_s \sin(\varphi(\gamma_r))$. Hence, when $\omega_s = 0$, it holds that $a_s = v_d$.

The energy also depends on other drone's features like the number of rotors, and the payload weight. There is no simple formula to calculate the unit energy as a function of the relative wind [29], so we introduce the compass rose to discretize the relative winds and to tabulate the energy for them. In this way, fixing the type of the drone (i.e., octocopter, quadcopter), the drone's ground speed $v_d$, the payload $\kappa$, and the speed of the global wind $\omega_s$, we can pre-compute the energy to traverse 1m: one energy coefficient for each sector of the compass rose (e.g., see Table 1.2 in Section 1.6). For the sake of simplicity, when all the involved parameters are clear from the contexts, the *energy* $E(\vec{AB})$ will denote the energy spent by a drone to fly the ground segment $\overline{AB}$ from $A$ to $B$.

### 1.3.2 The Compass Rose

In general, weather stations store the different winds detected by applying a discretization. In other words, given a *compass rose*, the winds are grouped in a finite number of sectors. To define

the sectors and the compass rose, we divide the turn angle at $O$ of a conventional Cartesian coordinate system $xOy$ into $4t$ sectors, each of width $\sigma = 180/2t$, where $t$ is the *compass rose cardinality*. Therefore, $\sigma$ defines the width of each sector. Conventionally, a *sector* $S_i$ contains the winds whose relative wind direction verify:

$$\left(i\frac{180}{2t}\ ;\ (i+1)\frac{180}{2t}\right] \quad 0 \leqslant i \leqslant 4t-1. \tag{1.2}$$

A *representative wind direction* $\rho_i$, with $0 \leqslant i \leqslant 4t-1$, is associated to each sector $S_i$. The representative $\rho_i$ is then used to compute the energy consumption of any relative wind direction that falls in $S_i$. The unit energy consumption for any wind in $S_i$ is denoted as $\mu_i$. Basically, the compass rose is used to bound to $4t$ the number of relative winds and thus the possible energy levels, as we will see in the next section. Table 1.2 reports the values of the energy required by a octocopter with different payloads for the 12 winds of the compass rose used in our experiments when $\omega_s = 10$m/s [29].

Having defined the energy consumption, we formally introduce the Minimum-energy Drone-trajectory Problem and explain how the drone can minimize the energy consumption.

## 1.4 The Problem Definition

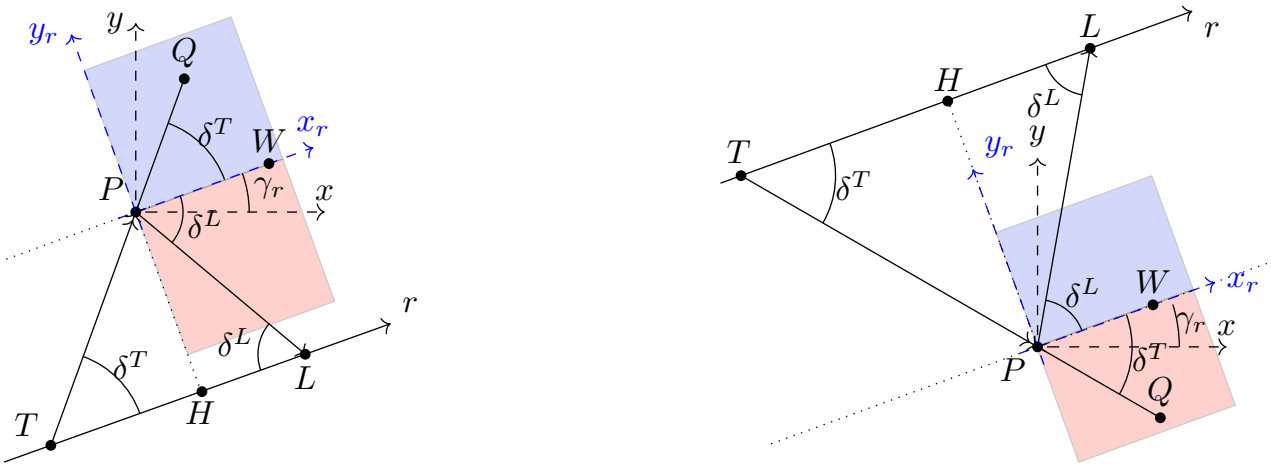
We assume the *truck-drone tandem delivery system* which consists of a *truck* that carries drones in the delivery area. When the global wind is absent, i.e., $\omega_s = 0$, the speed of the relative wind is zero and the energy spent by the drone does not depend on the direction of the trajectory. Hence, in this case, the best solution for the drone is to select the shortest trajectory from the truck's route to the customer. In presence of wind, the less consuming trajectory is the one parallel to the wind (see Eq. (1.1)). However, there may be no trajectory parallel to the wind that starts from the truck's path and reaches the customer. In that case, the selection of the drone's trajectory that minimizes the energy is not trivial. In the rest of the section, we first describe the problem, then we introduce our solution when the truck's path is a single line.

### 1.4.1 The Minimum-energy Drone-trajectory Problem (MDP)

Let $r$ be the line that represents the truck's path. So, the truck's path is represented by a vector $\vec{r}$ whose direction is $\gamma_r$. Notice that, in this chapter we do not consider the truck's speed. Let us consider a delivery point $P$ to be served by the drone. To serve $P$ from the road $r$ (see Figure 1.1), the truck selects the *take-off point* and *landing point* for the drone, called $T$ and $L$, respectively. Thus, the drone detaches from the truck at $T$ and flies on the line from $T$ to $P$ carrying the

payload. After the delivery, the drone (without payload) moves back on the line from $P$ to $L$, rendezvousing at the truck again. We select the trajectories making this assumption: the drone and the truck always move forward in the sense that the projection $H$ of $P$ onto the truck's path must follow $T$ and precede $L$.

Our problem is to find the optimal composed trajectory $\vec{TP} \cup \vec{PL}$ such that the energy consumed by the drone to serve $P$ is minimized under the assumption that $T \preccurlyeq H \preccurlyeq L$. We assume a constant wind during the whole delivery. We call this problem Minimum-energy Drone-trajectory Problem (briefly, MDP). In other words, we aim at minimizing the quantity $E(\vec{TP}) + E(\vec{PL})$. Changing $T$ and $L$, the slope (i.e., direction) of the segments $\overline{TP}$ and $\overline{PL}$ changes, and so, by the wind triangle rule, the relative winds $\varphi(\vec{TP})$ and $\varphi(\vec{PL})$, and their relative energy, change.




(a) $P \hookleftarrow r$.

(b) $P \hookrightarrow r$.

Figure 1.4: The relative winds on $\vec{TP}$ and $\vec{PL}$, where the quadrants $Q_1$ and $Q_4$ are highlighted in blue and red, respectively. A thorough explanation of the figure is reported in the text.

In the next, we show how to solve MDP by selecting the optimum trajectory among a discrete set of candidate trajectories.

### 1.4.2 Towards the MDP Solution

Considering $T$ and $L$ fixed on the track's route $r$, we calculate the relative wind on the trajectories $\vec{TP}$ and $\vec{PL}$ to find the energy consumption. First of all, we need to figure out whether $P$ is to the right or left of the truck. This can be quickly done by evaluating the sign of the determinant of the associated matrix $M = [v(\vec{TH}), v(\vec{TP})]$, where $v(\vec{TP})$ and $v(\vec{TH})$ are the column vectors[1]

[1]For instance, given $A = (1,1)$ and $B = (3,4)$, then $v(\vec{AB}) = \binom{2}{3}$.

associated to the drone's trajectories. Specifically, (i) $P$ *is on the left of* truck (denoted as $P \leftarrow r$) if $\det(A) > 0$; and (ii) $P$ *is on the right of* truck (denoted as $P \rightarrow r$) if $\det(A) < 0$.

In our discussion, we consider the Cartesian coordinate system $xPy$ of the mathematical wind, with origin in $P$. Let $H$ be the projection of $P$ on $r$ (see Figure 1.4). Let $\delta^T$ and $\delta^L$ be the angles formed, respectively, by the drone's trajectory $\vec{TP}$ and $\vec{PL}$ with the truck's route $r$. Namely, $\delta^T = \angle PTH$ and $\delta^L = \angle PLH$ can be easily computed as $\delta^T = \arctan(\overline{PH}/\overline{TH})$ and $\delta^L = \arctan(\overline{PH}/\overline{HL})$. Therefore, $\overline{TP} = \frac{\overline{PH}}{\sin(\delta^T)}$ and $\overline{PL} = \frac{\overline{PH}}{\sin(\delta^L)}$ (see Figure 1.4). So, it is evident that the length of the trajectories depends on the take-off or landing points and the slope of the truck's route $r$. Now, we can state the following.

**Theorem 1.** *The relative winds on $\vec{TP}$ and $\vec{PL}$ are:*

$$\begin{aligned} \varphi(\vec{TP}) &= \begin{cases} |\omega_d - (\gamma_r + \delta^T)|_{360} & \text{if } P \leftarrow r \\ |\omega_d - (\gamma_r - \delta^T)|_{360} & \text{if } P \rightarrow r \end{cases} \\ \varphi(\vec{PL}) &= \begin{cases} |\omega_d - (\gamma_r - \delta^L)|_{360} & \text{if } P \leftarrow r \\ |\omega_d - (\gamma_r + \delta^L)|_{360} & \text{if } P \rightarrow r \end{cases} \end{aligned} \tag{1.3}$$

*Proof.* We prove Eq. (1.3) when $P \leftarrow r$ (see Figure 1.4a). Consider the rotated Cartesian system $x_rPy_r$ with origin in $P$, $x_r$-axis parallel to $r$ and oriented in the same direction as $r$. Recall that the $y_r$-axis forms a 90 counter-clockwise angle with $x_r$. Note that the angle between $x$ and $x_r$ has width $\gamma_r$. Observed that $r$ and $x_r$ are parallel, the segment $\overline{TP}$ forms the same angle $\delta^T = \angle WPQ$ on $x_rPy_r$, and the angle $(\gamma_r + \delta^T)$ on $xPy$. Hence, the direction of the relative wind on $xPy$ is $\varphi(\vec{TP}) = \omega_d - (\gamma_r + \delta^T)$. Also the returning path $\vec{PL}$ forms the angle $-\delta^L$ on $x_rPy_r$. Hence, the direction of the relative wind on $xPy$ is $\varphi(\vec{PL}) = \omega_d - (\gamma_r - \delta^L)$.

The case where $P \rightarrow r$ is depicted in Figure 1.4b. Eq. (1.3) can be proven similarly to the left case. □

Hence, the relative wind depends on the wind direction, the direction of the truck's path, and take-off and landing points.

We now analyze in which sector of the compass rose both $\varphi(\vec{TP})$ and $\varphi(\vec{PL})$ fall, when $T$ and $L$ move on $r$. Let us discuss the case $P \leftarrow r$. When $T$ and $L$ move on $r$ towards $H$ (see Figure 1.4), it is easy to see that $\delta^T, \delta^L \in (0, 90]$. Precisely, $\delta^T = \delta^L = 90$ when $\overline{TH} = \overline{HL} = 0$, while $\delta^T$ and $\delta^L \to 0$ when both $\overline{TH}$ and $\overline{HL} \to \infty$. Then, the take-off and landing drone's trajectories $\vec{TP}$ and $\vec{PL}$ scan, respectively, the first quadrant $Q_1$ (shown in blue) and the fourth quadrant $Q_4$ (shown in red) of the Cartesian coordinate system (see Figure 1.4) with origin in $P$ and whose $x$-axis coincides with the relative wind $\varphi(\vec{r})$. Hence, observing that the relative wind on the truck's route is $\varphi(\vec{r}) = |\omega_d - \gamma_r|_{360}$, the take-off trajectories scan the compass rose starting from $\varphi(\vec{r})$

shown in blue in Figure 1.5, while the landing trajectories scan the compass rose starting from $\varphi(\vec{r})$ shown in red in Figure 1.5.

From that, Eq. (1.3) can be rewritten relatively to $\varphi(\vec{r})$:

$$\varphi(\vec{TP}) = \begin{cases} |\varphi(\vec{r}) - \delta^T|_{360} & \text{if } P \curvearrowleft r \quad Q_1 \\ |\varphi(\vec{r}) + \delta^T|_{360} & \text{if } P \curvearrowright r \quad Q_4 \end{cases}$$
$$\varphi(\vec{PL}) = \begin{cases} |\varphi(\vec{r}) + \delta^L|_{360} & \text{if } P \curvearrowleft r \quad Q_4 \\ |\varphi(\vec{r}) - \delta^L|_{360} & \text{if } P \curvearrowright r \quad Q_1 \end{cases} \tag{1.4}$$

As explained in Section 1.3.2, fixed $T$ and $P$, the relative winds $\varphi(\vec{TP})$ and $\varphi(\vec{PL})$ experienced by the drone are given by wind representative $\rho$ of the sector where they reside. The sectors visited when $T$ and $P$ move are listed starting from the sector $S_\tau$ where $\varphi(\vec{r})$ resides, i.e., $S_\tau$, where $\tau = \lfloor |\varphi(\vec{r}) - 1|_{360}/\sigma \rfloor$. Note that $\tau$ is computed from $\varphi(\vec{r}) - 1$ instead of $\varphi(\vec{r})$ because sectors in Eq. (1.2) are defined *left-open* and *right-closed*. By the previous Eqs. (1.4) and (1.2), it holds:

**Theorem 2.** *Let $\tau = \lfloor |\varphi(\vec{r}) - 1|_{360}/\sigma \rfloor$. When $\varphi(\vec{TP})$ or $\varphi(\vec{PL})$ scan $Q_1$, the drone spends the energy $\mu_i$ that depends on the winds $S_i$ of the compass rose whose indices $i$ are $\tau \leqslant i \leqslant |\tau + t|_{4t}$. When $\varphi(\vec{TP})$ or $\varphi(\vec{PL})$ scan $Q_4$, the drone spends the energy $\mu_i$ that depends on the winds $S_i$ of the compass rose whose indices $i$ are $|\tau - t|_{4t} \leqslant i \leqslant \tau$.*

Figure 1.5 shows two examples where $t = 3$ and so $\sigma = 30$ whose crossed quadrants are different in sectors and number. Specifically, when $\varphi(\vec{r}) = 0$, we cross the sectors $S_0$, $S_1$, and $S_2$ in $Q_1$, and $S_{11}$, $S_{10}$, and $S_9$ in $Q_4$, examining so exactly $t = 3$ sectors per quadrant. Instead, with $\varphi(\vec{r}) = 50$ we scan one more sector than before, crossing $S_1$, $S_2$, $S_3$, and $S_4$ in $Q_1$, and $S_1$, $S_0$, $S_{11}$, and $S_{10}$ in $Q_4$. Table 1.1 reports the sectors and the angles $\delta \in (0, 90]$ associated to each sector of $Q_1$ and $Q_4$ when $\varphi(\vec{r}) = 0$ and $\varphi(\vec{r}) = 50$ (see Figure 1.5). Generalizing, when $|\varphi(\vec{r})|_\sigma = 0$ we cross exactly $t$ sectors, instead when $|\varphi(\vec{r})|_\sigma \neq 0$ we scan $t + 1$ sectors. So, depending on which sector $T$ and $L$ fall in, the $\mu$ depends on the relative wind of the scanned sectors. Finally, note that changing $r$ or the direction of the global wind has the same effect: $\varphi(r)$ changes and so the candidate sectors change.

The take-off/landing positions that delimit the $i^{th}$ sector in $Q_1$ are those that form with $\gamma_r$ the $\delta_i$ angle so defined:

$$\delta_i = \begin{cases} \begin{cases} -|\varphi(\vec{r})|_\sigma + (i+1)\sigma & \text{if } i < t \\ 90 & \text{if } i = t \end{cases} & \text{if } |\varphi(\vec{r})|_\sigma \neq 0 \\ (i+1)\sigma & \text{if } |\varphi(\vec{r})|_\sigma = 0 \end{cases} \tag{1.5}$$

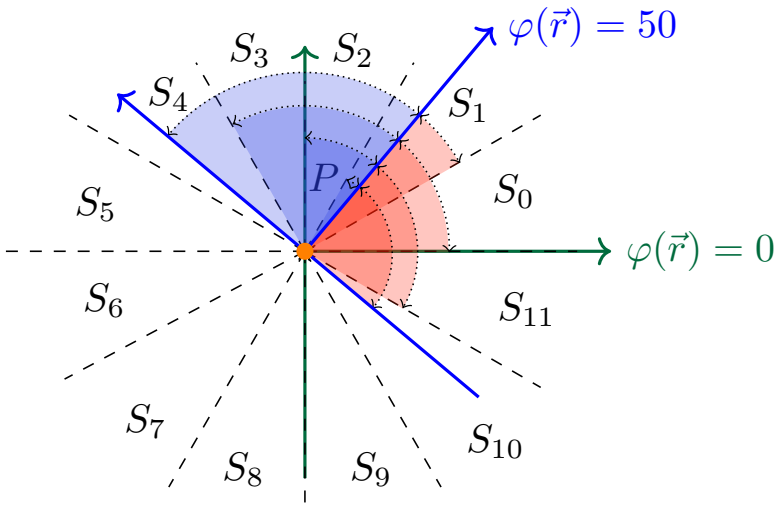



Figure 1.5: The candidate sectors: $|\varphi(\vec{r})|_\sigma \neq 0$ (blue) and $|\varphi(\vec{r})|_\sigma = 0$ (green).

Similarly, the take-off/landing positions that delimit the $i^{th}$ sector in $Q_4$ are those that form with $\gamma_r$ the $\delta_i$ angle so defined:

$$\delta_i = \begin{cases} \begin{cases} |\varphi(\vec{r})|_\sigma - 1 + i\sigma & \text{if } i < t \\ 90 & \text{if } i = t \end{cases} & \text{if } |\varphi(\vec{r})|_\sigma \neq 0 \\ (i+1)\sigma - 1 & \text{if } |\varphi(\vec{r})|_\sigma = 0 \end{cases} \tag{1.6}$$

In the following, we generally refer to the trajectory angles in Eqs. (1.5)–(1.6) with $\delta$ (omitting the index). We also use $\delta^T$ and $\delta^L$ when we generally refer to the angles of the take-off and the landing trajectories, respectively. These $\delta$ angles are very important because not only delimit the sector, but they also return the candidate trajectory in each sector. Indeed, the shortest trajectory in a sector forms the largest angle, which is $\delta$, with the truck's route. Note that $\delta$ depends on $\varphi(r)$: so the trajectories vary with the truck's route and the wind.

Table 1.1 shows the $\delta$ angles for the example in Figure 1.5.

To complete our analysis, it remains to find $T$ and $L$ given $\delta$. Given $P$ and a take-off point $T$ on the route path, the length of the drone's take-off route is $\frac{\overline{PH}}{\sin(\delta^T)}$, with $\delta^T \in (0, 90]$. Precisely, the drone selects $T$ such that $\delta_i^T$ is maximum varying $S_i$. The same holds for the landing points. So, at this point, we are ready to solve MDP. Given a global wind $\omega$, a truck's route $\vec{r}$, and a delivery point $P$, MDP aims at determining the *optimal wind sectors* $i^*$ and $j^*$ such that:

$$(i^*, j^*) = \arg\min_{i,j} \left\{ \frac{\mu_i^T}{\sin(\delta_i^T)} + \frac{\mu_j^L}{\sin(\delta_j^L)} \right\} \overline{HP} \tag{1.7}$$

where $i, j$ are the indices of the sectors given by Theorem 2, $\mu_i^T$ and $\mu_j^L$ are the energy consumption in these sectors $S_i$ and $S_j$ in $Q_1$ or $Q_4$, and $\overline{HP}/\sin(\delta_i^T)$ and $\overline{HP}/\sin(\delta_j^L)$ are the lengths of the

Table 1.1: The angles $\delta$ of the examples in Figure 1.5.

| $\varphi(\vec{r}) = 50$ | | | | | | $\varphi(\vec{r}) = 0$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_1$ | | | $Q_4$ | | | $Q_1$ | | | $Q_4$ | | |
| $\delta_i$ | $S_{\lvert\tau+i\rvert_{4t}}$ | $i$ | $\delta_j$ | $S_{\lvert\tau-j\rvert_{4t}}$ | $j$ | $\delta_i$ | $S_{\lvert\tau+i\rvert_{4t}}$ | $i$ | $\delta_j$ | $S_{\lvert\tau-j\rvert_{4t}}$ | $j$ |
| 10 | $S_1$ | 0 | 19 | $S_1$ | 0 | 30 | $S_0$ | 0 | 29 | $S_{11}$ | 0 |
| 40 | $S_2$ | 1 | 49 | $S_0$ | 1 | 60 | $S_1$ | 1 | 59 | $S_{10}$ | 1 |
| 70 | $S_3$ | 2 | 79 | $S_{11}$ | 2 | 90 | $S_2$ | 2 | 89 | $S_9$ | 2 |
| 90 | $S_4$ | 3 | 90 | $S_{10}$ | 3 | | | | | | |

trajectories candidates for being $\vec{TP}$ and $\vec{PL}$ in $S_i$ and $S_j$, respectively. Recall $\delta^T$ and $\delta^L$ are given by Eqs. (1.5) and (1.6) depending on the position of $P$ with respect to $r$. Note that, the length $\overline{HP}$ does not affect the minimum, and when $\delta^T = 90$, the minimum energy trajectory coincides with the shortest trajectory $\vec{HP}$ in absence of wind.

The SINGLE algorithm in Section 1.5 gives all the details to implement our solution.

Although the selection of the optimal trajectory for MDP is not affected by the distance of the customer from the truck's route, the energy changes with $\overline{PH}$. This means that the drone's battery might be depleted before reaching $P$. Nevertheless, if $P$ is not reachable with the minimum energy trajectory, it is also not reachable in absence of wind, i.e., using the trajectory $\vec{PH}$ of minimum distance $\overline{PH}$. While the best wind can be leveraged to reach customers unreachable in absence of wind. In the next section, we consider this aspect.

### 1.4.3 The Feasibility Analysis

Given that drones are energy-constrained vehicles with a limited battery capacity, their flight range is bounded. Let $B$ be the *battery budget* of the drone. By Eq. (1.7), for any feasible delivery point $P$, it holds:

$$B \geqslant \mu^T \frac{\overline{HP}}{\sin(\delta^T)} + \mu^L \frac{\overline{HP}}{\sin(\delta^L)}.$$

Fixed $\omega_s$, $v_d$, $\kappa$, and the truck's path $\vec{r}$, we compute the energy coefficients $\mu^T$ and $\mu^L$ that depend on the relative wind $\varphi(r)$ of the optimal trajectories of SINGLE, we have:

$$P_M(\varphi(r)) = \overline{HP} \leqslant \frac{B}{\frac{\mu^T}{\sin(\delta^T)} + \frac{\mu^L}{\sin(\delta^L)}} \tag{1.8}$$

i.e., $P_M(\varphi(r))$ represents the farthest distance reachable from the truck's route $\vec{r}$. By varying the wind direction and keeping all the other conditions the same, the farthest distance reachable from

$r$ varies because the relative wind changes, the compass rose is rotated, and the minimum energy trajectories change. Let $P_{\max} = \max_{\omega_d} P_M(\varphi(r))$ and $P_{\min} = \min_{\omega_d} P_M(\varphi(r))$ be the maximum and minimum distance that can be reached from $r$ with the minimum energy trajectory in presence of any wind, respectively. Moreover, let $P_{\text{st}}$ be the maximum distance reachable from $r$ in absence of wind. Note that $P_{\min} \leqslant P_{\text{st}} \leqslant P_{\max}$ for any wind because $P_{\text{st}}$ is one of the trajectories always tested in presence of wind to find the best trajectory (precisely, the one with $\delta = 90$). Any delivery closer than $P_{\min}$ is feasible with any wind (using either the minimum energy or the shortest trajectory), any delivery further than $P_{\max}$ is not-feasible with any wind (even if the best trajectory is implemented), while any delivery in between $P_{\min}$ and $P_{\max}$ is feasible only with the minimum energy trajectory and in the presence of some wind (i.e., the wind that returns $P_{\max}$). We use the comparison between $P_{\min}$ and $P_{\max}$ and the number of feasible points to evaluate the impact of our solution. The FEASIBILITY algorithm in Section 1.5 gives all the details.

## 1.5 The MDP Extension

In this section, we give the pseudocode for SINGLE discussed in Section 1.4, and then we propose the MULTI algorithm that applies MDP in the extended *multi-line scenario*. Eventually, we devise an algorithm, called FEASIBILITY, for determining "a priori" the feasibility of sending a drone for a delivery $P$ given a drone's budget $B$ using SINGLE or MULTI.

In Figure 1.6, we report the flowchart of the SINGLE, MULTI, and FEASIBILITY algorithms.

### 1.5.1 The SINGLE Algorithm

The SINGLE algorithm optimally solves MDP when $T$ and $L$ are selected from a single road, requiring $\mathcal{O}(t)$ time and space, where $t$ is the number of sectors for each quadrant. SINGLE has as input the wind vector $(\omega_d, \omega_s)$, the drone's speed $v_d$, the payload weight $\kappa$, the truck's road $\vec{r}$, the cardinality of sectors $t$ in the compass rose, and the delivery point $P$. The pseudocode of SINGLE is sketched in Algorithm 1.

The SINGLE algorithm works as follows. Initially (Line 1), we perform a pre-processing phase to compute the unitary energy costs according to the payload weight. Precisely, with $0 \leqslant i \leqslant 4t-1$, $\mu_i^T$ represents the energy spent for traversing a unitary distance when the global wind has speed $\omega_s$, the drone moves in $S_i$ with speed $v_d$ and carries the payload $\kappa$, instead, $\mu_i^L$, is the same as $\mu_i^T$ but without payload. Then, we compute the relative wind $\varphi(\vec{r})$ on the road $r$ according to the current global wind condition $\omega$ (Line 2), and also the wind sector with index $\tau$ (Line 3). At this stage, by exploiting Eq. (1.4) we identify the quadrants for $T$ and $L$ to scan in order to

**Algorithm 1:** SINGLE $(P, \omega, t, \vec{r}, v_d, \kappa)$

1 $\mu_0^T, \ldots, \mu_{4t-1}^T, \mu_0^L, \ldots, \mu_{4t-1}^L \leftarrow$ UNIT_COSTS$(\omega)$
2 $\varphi(\vec{r}) \leftarrow |\omega_d - \gamma_r|_{360}$
3 $\tau \leftarrow \lfloor |\varphi(\vec{r}) - 1|_{360}/\sigma \rfloor$
4 **if** $P \curvearrowleft r$ **then**
5 $\quad i^* \leftarrow \arg\min_{0 \leqslant i \leqslant t} \left\{ \frac{\mu^T_{|\tau+i|_{4t}}}{\sin\left(\delta^T_{|\tau+i|_{4t}}\right)} \right\}$
6 $\quad j^* \leftarrow \arg\min_{0 \leqslant j \leqslant t} \left\{ \frac{\mu^L_{|\tau-j|_{4t}}}{\sin\left(\delta^L_{|\tau-j|_{4t}}\right)} \right\}$
7 $\quad$ let $T$ such that $\overline{TH} = \overline{HP}\cot(\delta^T_{|\tau+i^*|_{4t}})$
8 $\quad$ let $L$ such that $\overline{HL} = \overline{HP}\cot(\delta^L_{|\tau-j^*|_{4t}})$
9 **if** $P \curvearrowright r$ **then**
10 $\quad i^* \leftarrow \arg\min_{0 \leqslant i \leqslant t} \left\{ \frac{\mu^T_{|\tau-i|_{4t}}}{\sin\left(\delta^T_{|\tau-i|_{4t}}\right)} \right\}$
11 $\quad j^* \leftarrow \arg\min_{0 \leqslant j \leqslant t} \left\{ \frac{\mu^L_{|\tau+j|_{4t}}}{\sin\left(\delta^L_{|\tau+j|_{4t}}\right)} \right\}$
12 $\quad$ let $T$ such that $\overline{TH} = \overline{HP}\cot(\delta^T_{|\tau-i^*|_{4t}})$
13 $\quad$ let $L$ such that $\overline{HL} = \overline{HP}\cot(\delta^L_{|\tau+j^*|_{4t}})$
14 **return** $T, L$

find the sub-routes with minimum energy. In other words, we find the best indices that return the optimal sectors for the take-off and landing sub-routes, i.e., $i^*$ and $j^*$, respectively, starting from $S_\tau$ (Lines 5–6). Now, depending on the position of $P$ with respect to the truck's road $r$, we can compute the optimal pair of points. If $P \curvearrowleft r$ (Line 4), we calculate the best $T$ searching in $Q_1$ and the best $L$ in $Q_4$, otherwise, if $P \curvearrowright r$ (Line 9), we do the opposite reasoning. Eventually, we return the best pair $T$ and $L$ (Line 14).

### 1.5.2 The MULTI Algorithm

The MDP extension to multi-line assumes that the truck's path is a polygon. The sides of the polygon determine the set $\Pi$ of different routes on which to run the SINGLE algorithm. In this case, we need to select not only the take-off and landing points, but also the side of the polygon where $T$ and $L$ reside. Moreover, $T$ and $L$ can be selected on two distinct sides. The pseudocode of MULTI is sketched in Algorithm 2.

Firstly, we build the two sets $\mathcal{T}$ and $\mathcal{L}$ which contain the set of feasible take-off and landing

**Algorithm 2:** MULTI $(P, \omega, t, \Pi, v_d, \kappa)$

```
1  T ← ∅, L ← ∅
2  foreach π ∈ Π do
3      {T, L} ← SINGLE (P, ω, t, π)
4      T ← T ∪ T, L ← L ∪ L
5  T ← −∞, L ← +∞
6  for i ∈ 0, ..., |T| do
7      for k ∈ i, ..., |L| do
8          if E(T[i]P) + E(PL[j]) < E(TP) + E(PL) then
9              T ← T[i], L ← L[j]
10 return T, L
```

points computed for each route $\pi \in \Pi$ (Algorithm 2, Line 1–4). After this, we compute the optimal solution by searching among all the possible pairs $(T, L)$ in $\mathcal{T}$ and $\mathcal{L}$, paying attention that $T$ precedes $L$ (Line 6–8). The current best pair is then saved (Line 9) and eventually returned (Line 10). The algorithm requires $\mathcal{O}(|\Pi|^2 + |\Pi|t)$ time.

### 1.5.3 The FEASIBILITY Algorithm

The FEASIBILITY algorithm finds the two thresholds, i.e., $P_{\max}$ and $P_{\min}$, used to establish the feasibility of a delivery $P$ regardless of the wind conditions for a given drone's battery budget. The algorithm considers all the winds $\omega_d \in \Omega$[2]. It takes $\mathcal{O}(|\Omega|t) = \mathcal{O}(t)$ time and $\mathcal{O}(1)$ space, where $t$ is the number of sectors of the compass rose. The pseudocode of FEASIBILITY is sketched in Algorithm 3.

Note that varying $\omega_d \in \Omega$ also varies the relative wind, but not the energy coefficients $\mu_0^T, \ldots, \mu_{4t-1}^T, \mu_0^L, \ldots, \mu_{4t-1}^L$. Only different sectors of the compass rose will be considered for different relative winds, as explained before. In our experiments, FEASIBILITY is then extended considering not only the truck's path $\vec{r}$ but all the lines $\Pi$ of the sides of the truck's polygon path. Note that changing $r$, although the energy coefficient of the compass rose are the same, the trajectory angles will change, as explained before.

[2]This is implemented by varying $\omega_d$ with regular steps between 0 to 360.

**Algorithm 3:** FEASIBILITY $(P, \omega, t, r, v_d, \kappa)$

$\mu_0^T, \ldots, \mu_{4t-1}^T, \mu_0^L, \ldots, \mu_{4t-1}^L \leftarrow$UNIT_COSTS$(\omega)$

$\overline{HP_{\max}} \leftarrow 0, \overline{HP_{\min}} \leftarrow +\infty$

**foreach** $\omega_d \in \Omega$ **do**

  $\varphi(\vec{r}) \leftarrow |\omega_d - \gamma_r|_{360}, \tau \leftarrow \lfloor |\varphi(\vec{r}) - 1|_{360}/\sigma \rfloor$

  $\Delta_T^* \leftarrow \min_{0 \leqslant j \leqslant t} \left\{ \frac{\mu^T_{|\tau - j|_{4t}}}{\sin\left(\delta^T_{|\tau - j|_{4t}}\right)} \right\}$

  $\Delta_L^* \leftarrow \min_{0 \leqslant j \leqslant t} \left\{ \frac{\mu^L_{|\tau + j|_{4t}}}{\sin\left(\delta^L_{|\tau + j|_{4t}}\right)} \right\}$

  $\overline{HP_M(\omega_d)} \leftarrow \frac{B}{\Delta_T^* + \Delta_L^*}$

  **if** $\overline{HP_M(\omega_d)} > \overline{HP_{max}}$ **then** $\overline{HP_{\max}} \leftarrow \overline{HP_M(\omega_d)}$

  **if** $\overline{HP_M(\omega_d)} < \overline{HP_{min}}$ **then** $\overline{HP_{\min}} \leftarrow \overline{HP_M(\omega_d)}$

**return** $P_{max}, P_{min}$

## 1.6 Performance Evaluation

In this section, we evaluate the performance of our algorithms when solving MDP. Specifically, we perform two kinds of evaluation: *numerical* in Section 1.6.2, and *simulation* in Section 1.6.3. In the previous, we simply run our proposed algorithms by varying all the aforementioned parameters, and by relying on our ad-hoc and coded environment. In the latter, instead, we rely on the open air simulator called BlueSky [33].

### 1.6.1 Settings and Parameters

We implemented our algorithms in Python language version 3.7, and run all the instances on an Intel i7-10genK computer with 16GB of RAM[3]. We consider a delivery area that consists of a circle of radius 5km whose origin coincides with the Cartesian coordinate system. When considering the *single line* (SL) scenario, we simply generate a line inside the area. In this case, the truck travels on a straight line (e.g., a highway), and deliveries take place in the area in front of such line. On the other hand, when we consider the *multi-line* (ML) scenario, we randomly generate a convex polygon with 8 sides that represent the closed path (route) of the truck to/from the depot. In other words, there are 8 contiguous roads that the truck traverses in the delivery area. By setting the starting point of the truck's route as one of the vertices of the polygon, the route follows the sequence of vertices traveled in a clockwise direction. In this case, the truck circumnavigates the

[3]The code is available on GitHub here: https://github.com/TheAnswer96/T-ITS23_bluesky-plugin

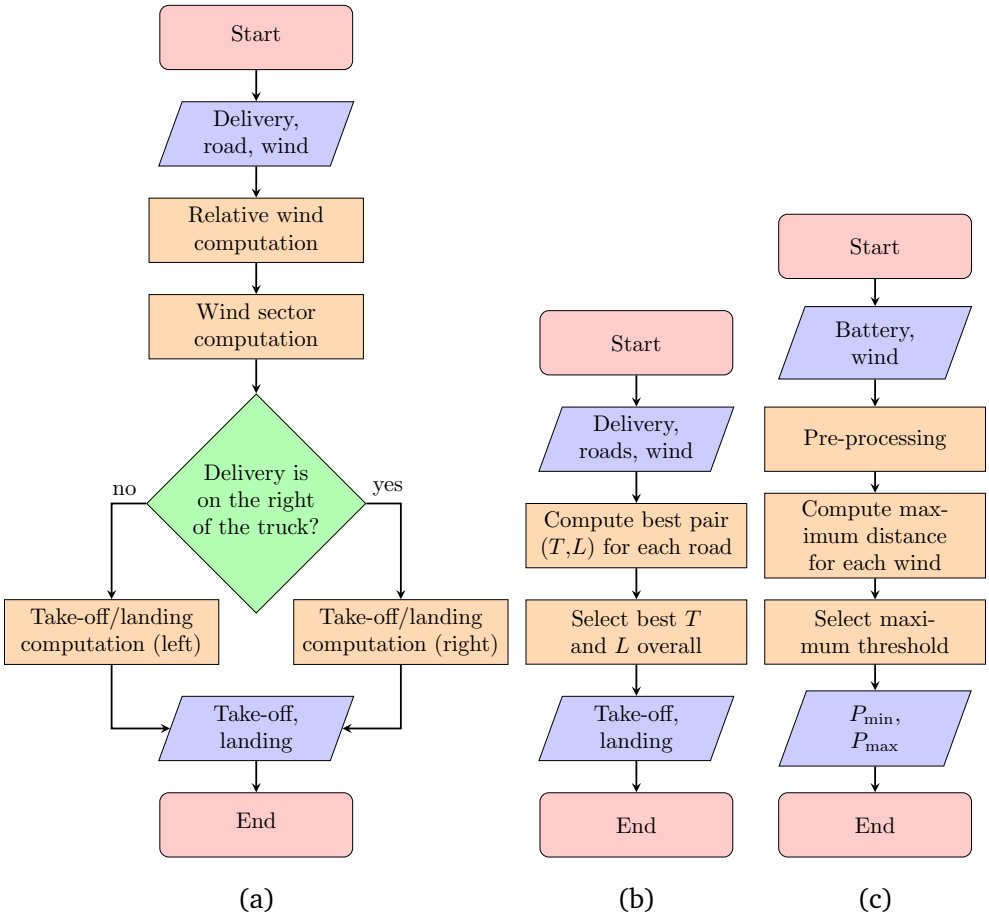



Figure 1.6: Flowchart of SINGLE (a), MULTI (b), and FEASIBILITY (c).

area where the deliveries occur. Also, $n = 20$ deliveries $P_1, \ldots, P_{20}$ are uniformly generated inside the delivery area.

We also consider two types of winds in the evaluation: *exhaustive winds* (EW) and *real winds* (RW). In EW, we vary $\omega_d$ with regular steps and consequently the relative wind $\varphi(\vec{r})$ on $r$ varies. In RW, the wind varies according to the collected data at different hours of the day in two locations in Corsica (France) during winter days. In general, such a region is characterized by low-intensity inland wind and high-intensity sea wind. When considering the EW scenario, we vary the wind speed $\omega_s = \{10, 20\}$m/s and the wind direction which varies according to a step of 15°. Moreover, we set the wind cardinality to $t = 3$. Concerning the parameters of the drone, we set the speed of the drone $v_d = \{10, 20, 30\}$m/s, the weight of the payload $\kappa = \{2, 6\}$kg, and the battery budget $B = \{2.5, 5, 10\}$MJ when dealing with the feasibility.

Also, as a reference, we compare our solution (minimum energy trajectory) with respect to the shortest trajectory, on both the single line (SINGLE-ST) and multi-line (MULTI-ST) scenarios, where ST denotes "shortest trajectory" (i.e., the one with $\delta = 90$) that is optimal when the wind

is absent. So, our numerical evaluations compare the energy required in presence of wind with the energy required in absence of wind.

### 1.6.2 Numerical Evaluation

In this section, we perform a numerical evaluation concerning the proposed algorithms. We will consider the compass rose formed by 12 sectors and hence $\sigma = 30$ (case $t = 3$). In Table 1.2, we report the different values of $\mu_i$ that have been used in the experiments. We tabulated these values since there is no simple formula to calculate them [29].

Table 1.2: Octocopter Energy consumption $\mu_i$ fixing $v_d = 10\text{m/s}$ and $\omega_s = 10\text{m/s}$, and with different payloads $\kappa$.

| | payload $\kappa$ | | | | payload $\kappa$ | | |
|---|---|---|---|---|---|---|---|
| $S_i$ | 0 | 2 | 6 | $S_i$ | 0 | 2 | 6 |
| 0 | 0.123 | 0.151 | 0.212 | 6 | 0.567 | 0.602 | 0.677 |
| 1 | 0.148 | 0.177 | 0.239 | 7 | 0.532 | 0.567 | 0.640 |
| 2 | 0.221 | 0.251 | 0.316 | 8 | 0.442 | 0.474 | 0.545 |
| 3 | 0.446 | 0.478 | 0.549 | 9 | 0.218 | 0.247 | 0.313 |
| 4 | 0.534 | 0.569 | 0.642 | 10 | 0.147 | 0.175 | 0.238 |
| 5 | 0.567 | 0.602 | 0.677 | 11 | 0.123 | 0.151 | 0.212 |

#### Single Line Exhaustive Winds Scenario

In Figure 1.7, we focus on the SL-EW scenario. In the $x$-axis we report the scanned wind directions $\omega_d$, while in the $y$-axis we report the ratio among the energy required for performing a delivery $P$ in absence of wind (by invoking SINGLE-ST), and the optimal energy required in presence of wind (by invoking SINGLE).

In the first row of Figure 1.7 we initially see that by fixing a configuration of the speed and payload of the drone, and varying the direction of the truck's route $\gamma_r$, the results are the same. In fact, they have just shifted by 180°. So, changing the direction of the road, the compass rose just rotates. Namely, for instance, the values on the $y$-axis when $\omega_d = 45$ and $\gamma_r = 0$ are exactly the same when $\omega_d = 45 + 180$ and $\gamma_r = 180$. Also the lengths of the minimum trajectories are the same because the $\delta$ angles coincide being $|\varphi(0)|_{30} = |\varphi(180)|_{30} = 0$. Focusing on the first plot with $\gamma_r = 0$, we can see that when the wind $\omega_d \in [90, 270]$, the ratio is 1, which means that

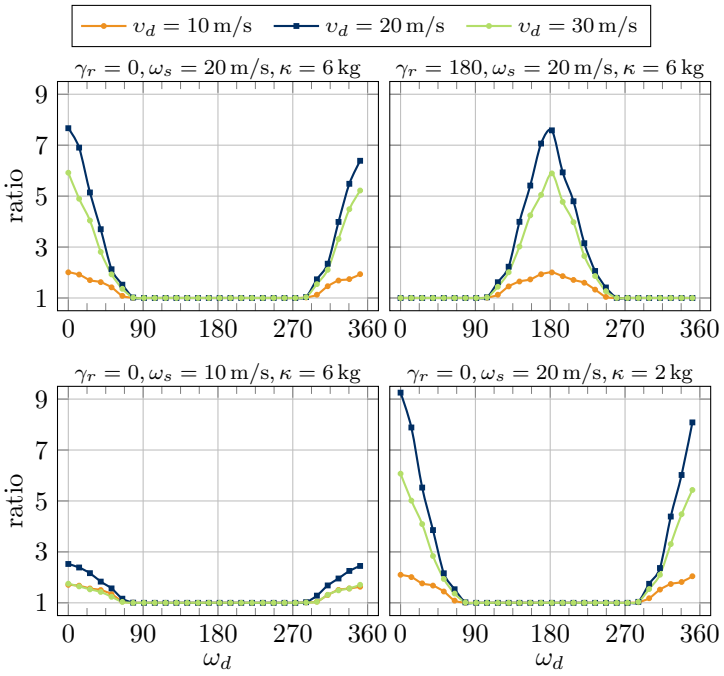



Figure 1.7: SL-EW scenario: SINGLE vs SINGLE-ST.

the two solutions work the same. In other words, the best solution provided by SINGLE forces the drone to fly perpendicularly like SINGLE-ST. Eventually, if there is a trajectory that starts from $r$ and reaches $P$ which is "more" tailwind (i.e., parallel or almost parallel to the global wind), SINGLE finds much better solutions with respect to the shortest perpendicular ones. In particular, it is interesting to see how much energy can be saved when $v_d$ varies in these circumstances.

In Figure 1.7, where $\omega_s = 20\text{m/s}$, when $v_d = 20\text{m/s}$ the minimum trajectory consumes $1/8$ of the energy in absence of wind, when $v_d = 30\text{m/s}$, $1/6$, and when $v_d = 10\text{m/s}$, $1/2$. This behavior is expected and comes from the adopted drone's energy reported for completeness in Section 1.A: the energy does not change linearly with $v_d$ as it seems at first glance in Eq. 1.9. Indeed, the energy gain also depends on the air-speed $a_s$ and as such on the relative difference between $v_d$ and $\omega_s$ as reported in Eq. (1.1). The gain is emphasized when such difference is positive, and it is reduced when such difference is negative. When we decrease the wind speed $\omega_s$ from 20m/s to 10m/s (i.e., plots of the first column), the energy saved is less because the head/tail component decreases. Finally, for the payload weight (i.e., comparing $\kappa = 6\text{kg}$ and $\kappa = 2\text{kg}$), the energy saved is more when the payload is lighter.

### Number of Sectors Analysis

In Figure 1.8, we analyze how the number of sectors can impact on the performance.

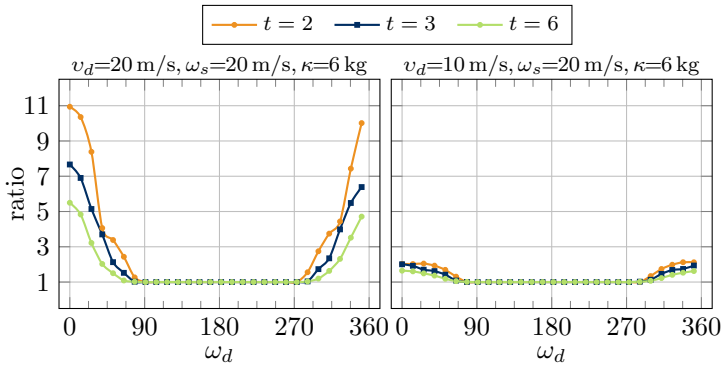



Figure 1.8: SL-EW scenario: comparison between different number of sectors.

Both the $x$-axis and $y$-axis have the same meaning as those in Figure 1.7. We varied the value of $t$. With $t = 2$, we have 4 different classes and therefore sectors with a width of 45°; with $t = 3$, we have 6 different classes and sectors with a width of 30°; and finally, with $t = 6$, we have 12 different classes and sectors with a width of 15°.

The discretized angle (in more classes) makes the energy consumption calculation more accurate, and therefore the used representative angle is closer to the actual experienced angle by the drone (i.e., when $t = 6$). However, when there are more sectors to deal with, the running time for computing the correct angles in presence of wind increases accordingly. On the other hand, when the number of classes is small, the energy consumption calculation is less accurate. So, for this reason, we decided to use a good trade-off between performance and complexity, relying on the average case with $t = 3$.

### Single Line Real Winds Scenario

In Figure 1.9, we focus on the SL-RW scenario by using real data of winds collected in a few locations in Corsica, France. In the $x$-axis we report the time (hours) of a winter day, while in the $y$-axis we report the same ratio as already explained in Figure 1.7.

We considered two cities that are differently affected by the wind, i.e., Cape Corse and Corte, during a whole day, from hour 1 to hour 24, during the winter season. Cape Corse is close to the sea where the wind strongly blows, while Corte is an inland city, and therefore the wind is much more moderate. In the first row of Figure 1.9 we report the results when the payload is 2kg, while in the second one when the payload is 6kg. The most interesting thing to observe is the impact of the wind in the first two-thirds of the day in Cape Corse. So, due to the strong wind in Cape Corse, the drone took advantage of the wind in its favor thus saving a lot of energy.

A counter-intuitive but correct behavior can be observed in Cape Corse when the drone's

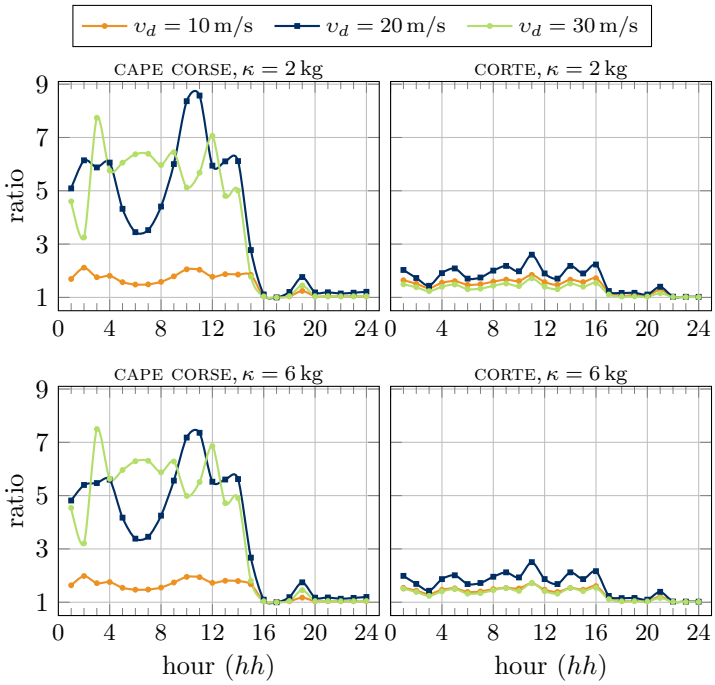



Figure 1.9: SL-RW scenario: SINGLE vs SINGLE-ST.

speed is the highest, i.e., $v_d = 30\text{m/s}$. In the previous Figure 1.7 we had seen that when the drone flies very fast, the energy saved is less. However, in Cape Corse the recorded wind from 4AM to 8AM was severely strong (even more than $30\text{m/s} \approx 110\text{km/h}$) and the drone, although flying at $v_d = 30\text{m/s}$, was able to save more energy than when it flies at the "most suitable" speed $v_d = 20\text{m/s}$. So, the energy saving depends on the relationship of the wind and drone's speed, and not only on drone's speed. Namely, the tailwind component is a function of $\omega_s$ in Eq. (1.1).

In Corte, however, since the wind was almost absent, and therefore the unitary energy consumption was almost the same for all the wind sectors, SINGLE prefers shorter routes, if not the shortest, to save energy. Comparing the energy savings of the same experimental runs with different weights reveals a predictable but subtle trend. In particular, we note that carrying a 6kg payload has a negative impact on energy consumption, and compared to carrying a 4kg payload, we receive less energy savings from the optimal route.

### Multi-line Scenario

In Figure 1.10, we focus on the ML case on both EW (first row) and RW (second row) scenarios. On each plot, in the $x$-axis we report the $i^{th}$ randomly generated delivery point $P_i$. The plots to the left report the energy, while the ones to the right report the length of the routes. The used parameters are $\omega_s = 20\text{m/s}, \kappa = 6\text{kg}$.

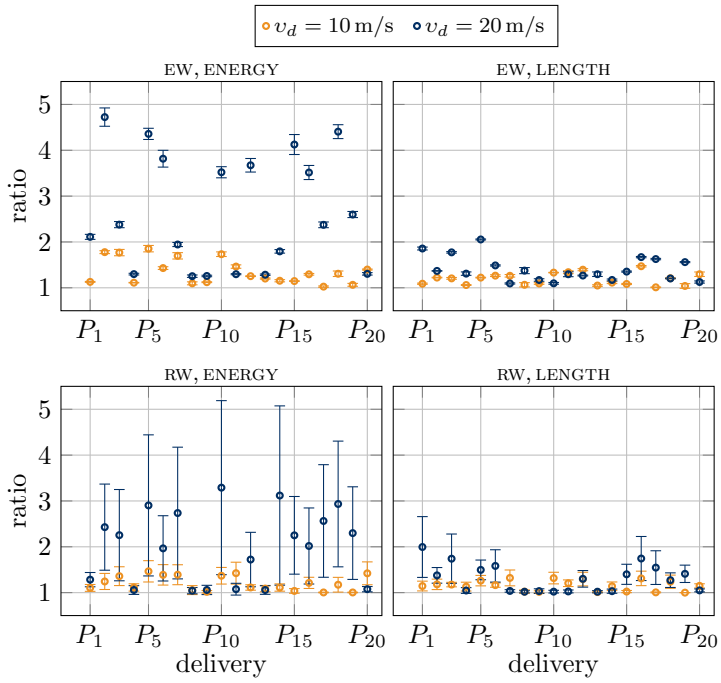



Figure 1.10: SL-EW/RW scenarios: MULTI vs MULTI-ST.

In Figure 1.10, when considering the ML-EW scenario (first row) with all the winds $\omega_d$, in the $y$-axis we report the *average* ratio (along with the 95% confidence interval) of the performance between MULTI-ST and MULTI in terms of energy (left plot) and length of paths (right plot). We can observe that on average, the energy saving of the 20 deliveries, when the drone's speed $v_d = 10\mathrm{m/s}$, is pretty stable. The distribution of the deliveries shows that the ratio is between 1 and 2, which means that, considering all the winds, the shortest trajectory is not that bad. However, when the drone's speed increases to $v_d = 20\mathrm{m/s}$, the same distribution of points is more variable. On average, the ratio is around 3, but many values are close to 5. In these cases, the wind helps in saving energy. About the length of the paths, the ratio is between the length of the paths of MULTI and the length of the paths of MULTI-ST. It is interesting to see that when $v_d = 20\mathrm{m/s}$, for the randomized delivery $P_5$ the energy saved was really large, and the length of the path is more than twice the shortest one. On average, for this speed, the paths are around 50% longer than the shortest ones. In Figure 1.10, when considering the RW scenario (second row), in the $y$-axis we report the same ratios just explained for the first row but considering all the real winds evaluated from 10 consecutive days in Corsica. The saving is high when $v_d = 20\mathrm{m/s}$. This trend also impacts the length of the paths (second row, right plot).

**Feasibility Analysis**

In Figures 1.11–1.12, we focus on the feasibility analysis in the SL/ML-EW scenario.

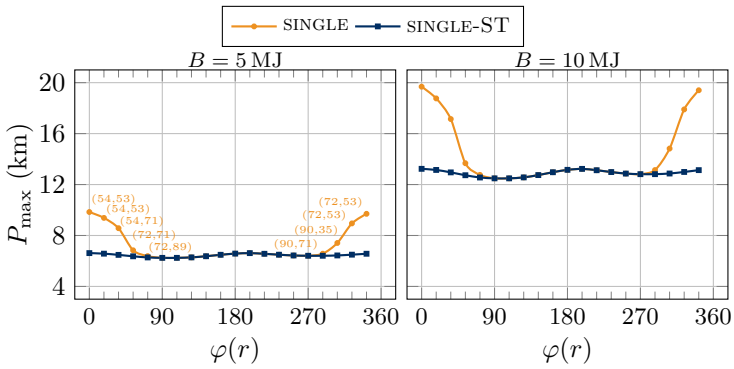



Figure 1.11: SL-EW scenario: furthest deliveries that can be served in presence of wind (SINGLE) and in absence of wind (SINGLE-ST) given $\varphi(r)$.

In Figure 1.11, we report the in the $x$-axis the relative wind of the truck $\varphi(r)$, while in the $y$-axis the value $P_{\max}$ which depends on the relative wind. Here, we consider $\omega_s = 10$m/s and $\kappa = 6$kg, while we vary the size of the battery 5MJ to the left, and 10MJ to the right. Not surprisingly, the values of $P_{\max}$ with twice the battery are doubled because having more energy available means flying more distance. About the case with 5MJ of energy, the farthest distance that the drone can reach is around 10km whose pair of take-off and landing angles is $(54, 53)$ when $\varphi(r) = 0$ (in orange). On the same relative wind, the drone could have gone at a maximum of 7km far away from the truck's road exploiting the shortest trajectory (in blue). We did not report the pairs of angles to the plot on the right since they match the ones on the left.

In Figure 1.12, we report the in the $x$-axis the *status* of the deliveries taking into account $P_{\max}$ and $P_{\min}$, while in the $y$-axis the number of deliveries for that particular status, in the ML-EW scenario. Here, we consider the size of the battery $B = 2.5$MJ to the left, and 5MJ to the right. A delivery can have one of the following status: *always feasible* (AF), if it is closer than $P_{\min}$; *always non-feasible* (AN), if it is farther than $P_{\max}$; *unknown* (U), otherwise. It is really interesting to observe the number of U and AF when comparing MULTI and MULTI-ST. In fact, when relying on MULTI it is possible to perform at least 35% of deliveries more than those done with MULTI-ST. Moreover, the number of non-feasible deliveries dramatically drops down when performing MULTI.

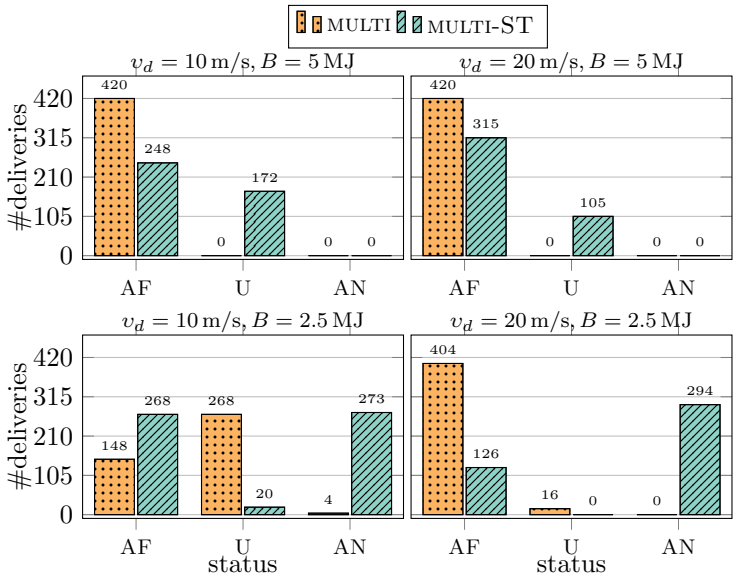



Figure 1.12: ML-EW scenario: status of deliveries.

### 1.6.3 Simulation Evaluation

For evaluating our algorithms on a simulated environment, we rely on BlueSky. BlueSky is an open Air Traffic Simulator (ATS), and is meant as a tool to perform research on Air Traffic Management and Air Traffic Flows [33]. At the best of our knowledge, BlueSky is the only simulator in which it is possible to simulate the flight of different aircraft by injecting the presence of the wind. Although BlueSky can model aircraft under the effect of the wind, there is not any available option that computes the actual energy consumption of them. So, we created an ad-hoc plug-in for BlueSky (still at a very preliminary stage though) that can estimate the energy consumed when a drone flies. When a flight simulation is done, BlueSky creates a log file (in CSV format) in which each line comprises different fields, like GPS coordinates, drone's speed, global wind, and so on. Basically, we parse the log file and for each two consecutive lines, we extract the two GPS coordinates, so that we can compute the actual flown distance and the drone's heading. Accordingly, knowing the wind experienced by the drone, we can then *precisely* estimate the energy consumption of it by using Eq. (10) in [29] (for completeness, reported in Eq. (1.10) in Section 1.A). Note that, the equation in [29] considers the current wind (i.e., $t = 90$) and so the energy consumption calculation is much accurate (and slower to compute) than our proposed model.

In Figure 1.13, we report the in the $x$-axis the wind direction, while in the $y$-axis the energy consumption per unit of distance (i.e., kJ/m). The energy per unit of distance is calculated through the ratio *total flown distance* to *total energy consumed*. Here, we have to specify two different cases: the results shown under the suffix "model" are determined exactly as done be-

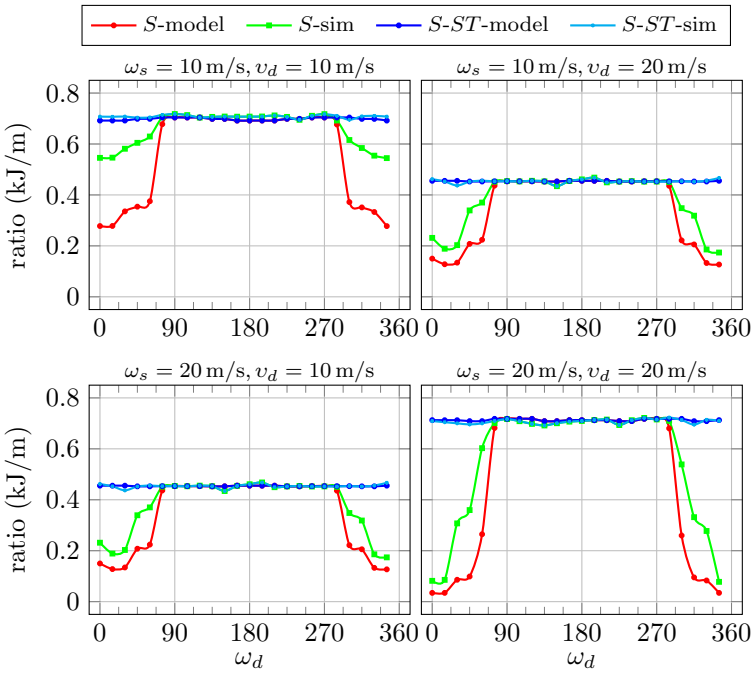



Figure 1.13: The simulated algorithms under BlueSky: $S$ stands for SINGLE.

fore in the numerical evaluation, while the ones under the suffix "sim" are determined inside the BlueSky simulator. The drone's altitude is 40m above the ground. We set the other parameters with respect to the settings in Section 1.6.1.

The simulated results confirm the numerical ones. The difference among the simulated and ideal perpendicular paths (SINGLE-ST model and sim) is almost zero, while slightly different is the situation with the non-perpendicular ones. We also observe that the largest energy consumption per meter occurs when the wind and the drone's speed match (i.e., $\omega_s = \upsilon_d$). In general, one can observe that the saved energy is less in the simulated environment than in the ideal numerical environment. Except the case with $\omega_s = \upsilon_d = 10$m/s, the gap in terms of required energy per unit of distance is $\leqslant 0.1$kJ/m in favor of the ideal numerical model. This is because the simulator uses more complicated trajectories for the aircraft while flying between two points. We have indeed observed that the drone does turning maneuvers when changing its direction.

## 1.7 Conclusions

For the first time, to the best of our knowledge, we adapt the trajectory of a drone to the wind. We consider a truck-drone tandem delivery system. The drone actively reacts to the wind by adopting the "most tailwind" available trajectory between the truck's path and the delivery. Two solutions are proposed for solving MDP, i.e., when the truck moves on a line (e.g., a highway)

in front of the deliveries, and when the truck moves on a polygon that delimits a convex area where the deliveries take place. Currently, our proposed model presents some limitations. For instance, if the wind dynamically changes during a delivery, the drone does not update its path in order to save energy. Moreover, the discretization in multiple wind classes determines approximated angles if we worked in the continuous space, and sometimes the energy estimations differ from the actual ones. Nevertheless, our proposed model has the undoubted value of showing the positive role that the wind can play in the drone flight. In addition, even though the computations of the FEASIBILITY method are static, it can predict the mission outcome ahead of time, independent of wind conditions, allowing the mission planner to decide whether to approve or deny a flight. Furthermore, while the wind can vary throughout a mission, a significant shift in both the wind's direction and speed is not particularly truthful. Indeed, as thoroughly investigated in Section 1.6.2, the majority of the delivery performed using the optimal route can be completed successfully regardless of the wind conditions. The small number of deliveries that are *unknown* or *always non-feasible* occurred under extremely unusual wind conditions that are difficult to encounter in daily life.

In the future, we would like to confirm our findings by using other simulators, and eventually extending our investigation to a test-bed made by off-the-shelf drones. We will also investigate, in a more detailed way, the circumstances, such as the time of day, in which a mission may fail, developing relying on our model robust algorithms able to react even to harsh wind gusts.

# Appendix

## 1.A Drone-Wind Energy Model

The motion of drones is regulated by physical properties [29]. The total required thrust is $\mathbb{T} = Wg + F_{\mathrm{D}}$, where $W$ is the total weight of the drone, $g$ is the gravitational constant, and $F_{\mathrm{D}} = \frac{1}{2}\rho\, a_s^2\, C_D A$ is the total drag force, where $\rho$ is the air density, $a_s$ is the drone's air speed (see Eq. (1.1)), $A = \pi R^2$ is the cross sectional area ($R$ is the rotor radius), and $C_D$ is the drag coefficient. Having computed $\mathbb{T}$, we estimate the required power $\mathbb{P}$ for a steady flight, which is $\mathbb{P} = \mathbb{T}(v_d \sin(\alpha) + s_i)$, where $\alpha = \arctan(F_{\mathrm{D}}/Wg)$ is the pitch angle, and $s_i$, which is the induced velocity required for a given thrust $\mathbb{T}$, can be obtained by solving the implicit equation [47] $s_i = s_h^2/\sqrt{(v_d\cos(\alpha))^2 + (v_d\sin(\alpha) + s_i)^2}$, where $s_h = \sqrt{\mathbb{T}/2\rho A}$ is the induced velocity at hover [47]. Note that, $F_{\mathrm{D}}$ and so $\alpha$ and $\mathbb{P}$ depend on the drone's direction $\gamma_r$. So, fixed $\omega = (\omega_d, \omega_s)$, the unitary energy $\mu(\gamma_r)$ of travel along a segment $r$ with direction $\gamma_r$ is calculated as follows:

$$\mu_\omega(\gamma_r) = \mathbb{P}/v_d. \tag{1.9}$$

Therefore, the energy consumed for traversing one edge $\gamma_r$ of length $\lambda(\gamma_r)$ can be expressed as:

$$d_\omega(\gamma_r) = \mu_\omega(\gamma_r)\lambda(\gamma_r). \tag{1.10}$$

# Chapter 2

# On the Scheduling of Conflictual Deliveries in a Last-mile Delivery Scenario with Truck-carried Drones

## 2.1 Introduction

In this chapter, we investigate the cooperation between a *truck* and one or multiple drones for last-mile package delivery. With the help of drones, delivery companies can perform more deliveries and hence extend their revenue and business [48]. In fact, drones can deliver small packages quickly as they can easily traverse difficult terrain [49], often using shorter routes not otherwise possible for trucks. In this scenario, the drones fly (starting from the truck) to the assigned locations and deliver the packages, then leave and meet again with the truck to perform new deliveries. However, significant challenges arise due to such constraints as the drones' high energy consumption and their *current* inability to simultaneously serve multiple customers at a time [50, 51]. The premise under consideration is that a delivery company has to make several deliveries to the customers in a city by relying on a truck carrying a fleet of drones with the same capabilities.

In this chapter, we study this problem by assuming that the main depot knows the locations of the customers to visit and the road to travel for the truck. Therefore, before leaving the depot, the delivery company plans the drones' flying sub-routes to accelerate the deliveries, considering not only the energy used by the drones but also the revenue generated. A sub-route is uniquely identified by a starting and a rendezvous point on the truck's route. Since drones have limited battery capacity, *a delivery has a cost* in terms of the energy required, and this limits the number of deliveries that a drone can perform. Also, *conflicts among deliveries* can occur if they cannot be

accomplished by the same drone, i.e., if the corresponding sub-routes, and hence their delivery intervals, intersect each other (see Figure 2.3.1a). In addition, *each delivery has an associated reward* that characterizes its importance (e.g., a larger reward means higher priority). So, given the truck's route in the city, *the goal is to plan a scheduling for the drones such that the total reward is maximized subject to the constraints of limited drone's battery and the conflicts among deliveries*.

### 2.1.1 Motivations

The existing works in this area mainly study the problem of scheduling deliveries without considering neither any budget constraint for the drones nor possible conflicts among deliveries [30, 52, 53]. The absence of such details makes it harder to model real-world scenarios. Moreover, the proposed solutions in drone-based delivery applications are often determined by heuristic or meta-heuristic algorithms which do not provide any guaranteed results, in terms of solution's goodness [36, 54]. In contrast, in this chapter, we incorporate the aforementioned constraints in a last-mile delivery scenario, while also providing optimal, approximation, and heuristic solutions for our problem.

Another interesting aspect concerns the fact that the truck's route is already given. It is reasonable to assume that we know the route of the truck in advance for various reasons. For example, there can be a truck's route that is efficient most of the time considering traffic, road condition, and safety, and by using such a route, truck drivers can conduct their everyday delivery tasks comfortably and safely. Given the truck's route, we envision increasing the number of performed deliveries by exploiting a fleet of drones and reaching additional customers that were not first considered during the route planning. Our objective is not to change the route of the truck to serve more customers, but rather to serve the largest number of customers in the neighborhood of the truck's route. Therefore, the emphasis of this chapter is to possibly increase the number of deliveries without modifying the truck's route.

The results of this chapter are summarized as follows:

- We define a novel optimization problem, called Scheduling Conflictual Deliveries Problem (SCDP), and prove it to be *NP*-hard. We also propose an Integer Linear Programming (ILP) formulation for SCDP.

- For the single drone case, we design optimal and approximation algorithms, while for the multiple drones case, we propose approximation and heuristic algorithms.

- We evaluate the performances of our algorithms on synthetic datasets.

The remainder of this chapter is structured as follows. Section 2.2 surveys the related works. Section 2.3 introduces the SCDP showing its *NP*-hardness, and proposes an ILP formulation. Section 2.4 presents algorithms for solving SCDP with a single drone, while Section 2.5 devises algorithms for solving SCDP with multiple drones. Section 2.6 evaluates the algorithms on synthetic datasets. Finally, Section 2.7 offers conclusions and future research directions.

## 2.2 Related Work

This section reviews the literature related to the problem of delivering packages with a truck carrying one or multiple drones, categorizing the works based on whether the deliveries are scheduled to a single or multiple drones. We recall that our proposed model considers both scenarios.

### 2.2.1 Single Drone Deliveries

The truck-drone cooperation in the last-mile delivery has been investigated for the first time when the flying sidekicks traveling salesman problem (FSTSP) has been introduced in [30]. The FSTSP is a particular case of the TSP, where drones take off from the truck, fly to deliver packages to customers, and go back to the truck in another place. Both vehicles can perform deliveries, but each has to stop waiting for the other at the rendezvous location. The authors propose a mixed-integer linear programming formulation (MILP) and two heuristic solutions for solving the FSTSP. Differently from us, they do not take into account any reward and conflicts among deliveries, also because the truck and the drone can do multiple trips to/from the depot.

In the work [52], a single drone carried by a truck performs all the required deliveries. Differently from us, once any delivery has been performed, a drone can immediately recharge its battery. This means that all deliveries are feasible and, accordingly, no scheduling is required.

The heuristic algorithm proposed in [36] initially focuses on the truck's route and then improves the solution by considering the possible drone's sub-flights. A TSP solution for the truck only that fixes the order in which all the customers are served, is at first computed. Then, short drone sub-routes are greedily created by excluding some customers from the truck's route in order to reduce the overall makespan (i.e., the time difference between the start and end of a delivery sequence). However, the truck or the drone has to wait for the other at the rendezvous positions. Also, as we do, the truck is not allowed to go back on its route.

In the delivery system in [55], a truck carries a drone to deliver packages to customers. The objective is to minimize the total tour duration to serve all the customers. Differently from us, here there can be incompatible customers for the drone (e.g., package too heavy, or customer too

far). So, the authors propose a MILP formulation based on timely synchronization of the truck and the drone trajectories. They also introduce a dynamic programming recursion based on an exact branch-and-price approach capable of optimally solving small instances.

### 2.2.2 Multiple Drones Deliveries

The investigated problem of delivering goods is further extended by considering also a fleet of drones, and numerous research works have been published.

In [54], the authors consider a scenario with multiple rechargeable drones. Drones can only carry one package at a time and have to return to the truck's roof to charge their battery after each delivery. The speed is the same for both drones and trucks, but their mobility metric is different (Euclidean metric for drones, Manhattan metric for the truck). The authors present a heuristic to solve the problem of finding a good schedule for the truck and all the drones that minimizes the average delivery time of the packages. In the chapter, we do not consider the replenishment of batteries for drones, and also we optimize the reward and not the average delivery time.

A similar work to ours is proposed in [37]. The authors consider a predefined truck's route, and the problem is to optimize the planning of drones to/from the truck while serving the customers. However, unlike us, they need to determine the launch and meeting points between drones and the truck, while in our investigation these points are given in input. Furthermore, these points are calculated in such a way that the intervals generated do not intersect with each other, thus making all deliveries conflict-free. Importantly, the authors assume that drones do not have battery constraints. While their goal is to reduce total makespan, our goal is to maximize the reward for deliveries.

The delivery with a truck and drones with a rechargeable station is presented in [53]. The station can recharge multiple drones, and it is located near customer areas and away from the main depot. After having determined the lower bound of the number of drones that the station can handle, the authors show that this problem is a combination of TSP and parallel identical machine scheduling problems. Through this approach, they successfully reduce the complexity of the problem and obtain an exact solution. Differently from us, their goal is to minimize the overall makespan, under the assumption that the deliveries are equally relevant (each delivery has the same reward), and that all the deliveries are conflict-free.

A hybrid truck-drones scenario is presented in [56], where the authors, differently from us, propose to simultaneously employ trucks, truck-carried drones, and independent drones to construct a more efficient truck-drone parcel delivery system. A novel routing and scheduling algorithm is proposed to solve the hybrid parcel delivery problem.

In [57] a clustering approach is proposed to find locations for a truck in which it stops and deploys drones to deliver packages to near locations. In these cases, the total delivery time is determined by the longest flight time among drones in each cluster. The authors aim at minimizing the total delivery time by having a truck moving among the centers of clusters as drones operate in each cluster. However, all the deliveries can be performed since conflicts among them are not contemplated.

Last but not least, the seminal FSTSP has been extended, by the same authors in [58], for multiple drones (mFSTSP). A MILP formulation only suitable for small inputs, as well as faster heuristics for any input, have been devised to reduce the overall makespan.

## 2.3 Problem Formulation

In this section, we first introduce the problem's underlying assumptions, then present the system and delivery models, and formally define the Scheduling Conflictual Deliveries Problem (SCDP). We also show that SCDP is *NP*-hard.

### 2.3.1 Problem Assumptions

In this chapter, we make a few problem assumptions listed as follows:

- The drones deliver a single package at a time due to stringent payload constraints. In other words, a drone cannot carry more than a single package.
- The route of the truck is already pre-planned and it is given. Therefore, the truck cannot dynamically make detours or modifications.
- Each drone has only a single battery. Although a drone can perform multiple single deliveries with the same battery, it cannot swap or recharge its battery for additional deliveries.

### 2.3.2 System and Delivery Model

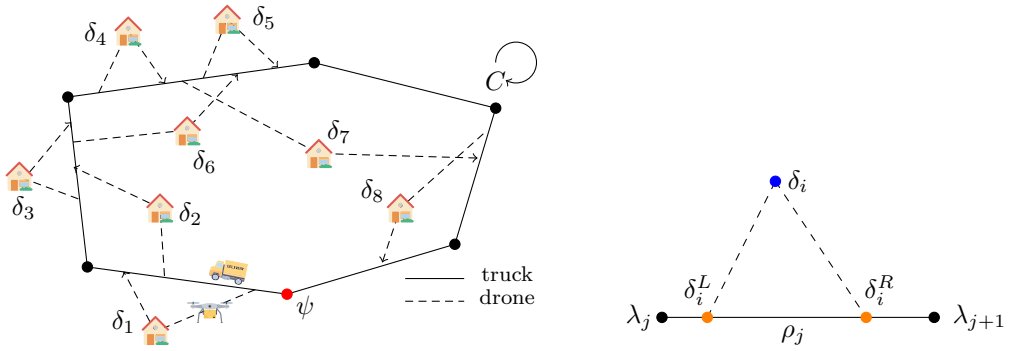
A drone's delivery is done by planning a sub-flight that passes through three points. Specifically, the drones take off (with packages) from the truck which continues to drive in the city, deliver packages to the customers, and return to the rendezvous locations with the truck again.

Let $A$ be the 2-D *delivery area* representing our last-mile delivery (application) scenario. Let $\psi \in A$ be the *depot* from where the deliveries start. The position of the depot is located at $(x_\psi, y_\psi)$, assumed to be at the origin of the Cartesian coordinate system in $(0,0)$. Such a delivery area also

comprises roads and customers' positions. At the depot, a *truck* is in charge of transporting a fleet of $m$ *drones* $d_1, \ldots, d_m$ with the same capabilities used for deliveries in $A$. The truck *does not* perform deliveries; it only carries the drones within $A$. Let $\rho_j$ be a straight road delimited by two endpoints $\lambda_j$ and $\lambda_{j+1}$, where $j = 0, \ldots, r$ and $\{\lambda_1, \ldots, \lambda_r\} \subset A$. The truck leaves $\psi$ visiting $\lambda_1$ along $\rho_0 = \overline{\psi\lambda_1}$, then $\lambda_2$ along $\rho_1 = \overline{\lambda_1\lambda_2}$, and so on, up to $\lambda_r$, and eventually going back to $\psi$. These segments make a polygon (*cycle*) $C$ formed by the sequence $\lambda_0, \lambda_1, \ldots, \lambda_r, \lambda_{r+1}$ such that $\lambda_0 = \lambda_{r+1} = \psi$. Let $D = \{\delta_1, \ldots, \delta_n\} \subset A$ be the set of $n$ distinct points or locations to be served (i.e., the *customers*) by the drones. Each customer's (delivery) point $\delta_i$ has a pair of coordinates $(x_{\delta_i}, y_{\delta_i}) \in A$, for $i = 1, \ldots, n$. Figure 2.3.1a illustrates the delivery area $A$ with roads and customers.

For each customer's location $\delta_i$, let $\delta_i^L$ and $\delta_i^R$ be respectively the *launch point* and *rendezvous point* of the drone on the truck's route. For any delivery $\delta_i$, both the $\delta_i^L$ and $\delta_i^R$ are already known and given in input. In the example in Figure 2.3.1b, the truck travels from $\lambda_j$ to $\delta_i^L$ carrying the drone, which in turn takes off at $\delta_i^L$ flying towards $\delta_i$ delivering the package, and finally continues flying towards $\delta_i^R$. In the meanwhile, the truck continues its route reaching other endpoints. When both the truck and the drone arrive at $\delta_i^R$, the former gathers again the drone and they continue to travel up to point $\lambda_{j+1}$. Note that $\delta_i^L$ and $\delta_i^R$ can lie on different roads. In general, $\delta_i^L$ lies on road $\rho_j$ while $\delta_i^R$ lies on road $\rho_z$, where $0 \leqslant j \leqslant z \leqslant r$ (see Figure 2.3.1a). Note that, since the truck's route is predefined, it is forbidden for the truck to travel back and forth for picking up the drones. This is our most important assumption which impacts the problem definition and depends on the fact that the truck is, in some sense, an opportunistic means of transportation for the drones. The truck has its own route due to other assigned tasks or because it is moving on a track, as it is common for emerging means of transportation with low greenhouse gas emissions. In any case, the truck cannot change its route.

Let $w_i \geqslant 0$ be the *energy cost* (or *weight*) for a drone in terms of energy spent to perform a single delivery $\delta_i$ flying to/from the truck. Although the $w_i$ associated with the delivery $\delta_i$ can be derived leveraging either the equations reported in [29] or in Chapter 1, it is worth pointing out that the algorithms provided in the following guarantee a fixed approximation regardless the energy cost distribution. Let $B \geqslant 0$ be the drone's *energy budget* in terms of battery capacity which limits the total number of feasible deliveries. A drone can perform multiple individual deliveries with a single battery of initial capacity $B$. Each delivery can be performed if it requires less energy than the residual energy budget. In other words, if the drone's current residual energy is not enough for additional flights, it is not possible to perform further deliveries unless a new battery is swapped. We do not consider the possibility to exchange the battery. We assume that




(a) (b)

Figure 2.3.1: An example delivery area $A$ with 6 roads and 8 customers $\delta_i$ to serve. The depot $\psi$ is at $(0,0)$; The truck's route is the solid line, while the drones' routes are the dashed lines (a); A road with a delivery $\delta_i$ to do: the launch $\delta_i^L$ and rendezvous $\delta_i^R$ points are highlighted (b).

drones have to rely only on one battery charge. We remark that, in the multiple drones case, each drone has its own battery of capacity $B$.

Let $p_i \geqslant 0$ be the *reward* for executing a delivery $\delta_i$. The reward characterizes *premium users* as having higher priority than *regular users*. For instance, delivery companies offer different subscriptions according to the following rule: "*The more you pay, the faster you receive your parcels*". In our context, the higher the priority of delivery $\delta_i$, the larger the reward value $p_i$. Hence, to satisfy the premium users, the scheduling for the drones needs to prioritize the deliveries, guaranteeing first the ones belonging to the premium users (because they have more reward) and second the ones from regular users.

Deliveries are characterized by *intervals* along the truck's route. For any delivery $\delta_i$, the launch and rendezvous points $\delta_i^L$ and $\delta_i^R$ define the drone's *delivery interval* $I_i = [\delta_i^L, \delta_i^R]$. We assume that the truck visits first $\delta_i^L$ and then $\delta_i^R$, so $\delta_i^L \preccurlyeq \delta_i^R$. Let $I = \{I_1, \ldots, I_n\}$ be the *interval set* associated with the deliveries, where $1 \leqslant i \leqslant n$. Two intervals $I_i$ and $I_j$ are said to be *compatible* if their intersection is empty, i.e., $I_i \cap I_j = \varnothing$, for $i \neq j$; otherwise, the two intervals are in *conflict*. In other words, $I_i$ and $I_j$ are compatible if $\delta_i^R \preccurlyeq \delta_j^L$ or $\delta_j^R \preccurlyeq \delta_i^L$. This is reasonable with the fact that if a drone is still performing a delivery, i.e., it is still in flight, it cannot start a new delivery without first reaching the truck. A subset $S \subseteq I$ is said to be *compatible* if $I_i \cap I_j = \varnothing$ for any pair $I_i, I_j \in S$. In other words, a drone can perform any subset of such deliveries as long as it has enough battery. Precisely, a given compatible $S \subseteq I$ is *feasible* if the energy cost $\mathcal{C}(S) = \sum_{I_i \in S} w_i \leqslant B$ (the energy budget). The *reward* of a feasible set $S$ is $\mathcal{P}(S) = \sum_{I_i \in S} p_i$.

Recall that $m$ is the number of drones. Two feasible subsets $S_p, S_q \subseteq I$ can be *assigned* to two drones $d_p$ and $d_q$, with $1 \leqslant p \neq q \leqslant m$, if $S_p \cap S_q = \varnothing$. Assuming that $S = \{S_1, \ldots, S_m\}$ consists of

$m$ feasible sets assigned to the drones $\{d_1, \ldots, d_m\}$, the *overall reward* $\mathcal{P}(S)$ is defined as the sum of the reward of each drone, i.e., $\mathcal{P}(S) = \sum_{j=1}^{m} \sum_{I_i \in S_j} p_i$.

Table 2.3.1 reports the adopted terminology in this chapter.

Table 2.3.1: Symbol description.

| **Symbol** | **Description** |
|---|---|
| $A$ | 2-D delivery area |
| $\psi$ | depot from where the deliveries start |
| $m$ | number of drones |
| $d_1, \ldots, d_m$ | ID of each drone |
| $\rho_j$ | straight road delimited by two endpoints |
| $\lambda_j$ | endpoint of a road |
| $C$ | cycle of the truck |
| $n$ | number of deliveries |
| $\delta_i$ | a delivery |
| $(x_{\delta_i}, y_{\delta_i})$ | location of a delivery |
| $D = \{\delta_1, \ldots, \delta_n\}$ | set of deliveries |
| $\delta_i^L$ | launch point of the drone on the truck's route for $\delta_i$ |
| $\delta_i^R$ | rendezvous point of the drone on the truck's route for $\delta_i$ |
| $w_i$ | energy cost for a drone to perform delivery $\delta_i$ |
| $B$ | energy budget for a drone for all the deliveries |
| $p_i$ | reward for executing a delivery $\delta_i$ |
| $I_i = [\delta_i^L, \delta_i^R]$ | delivery interval of a delivery $\delta_i$ |
| $I = \{I_1, \ldots, I_n\}$ | interval set associated with the deliveries |
| $\mathcal{C}(S)$ | energy cost function for a given subset of deliveries $S \subseteq I$ |
| $\mathcal{P}(S)$ | reward function for a given subset of deliveries $S \subseteq I$ |

### 2.3.3 Problem Definition

Let us now formally define the delivery scheduling problem.

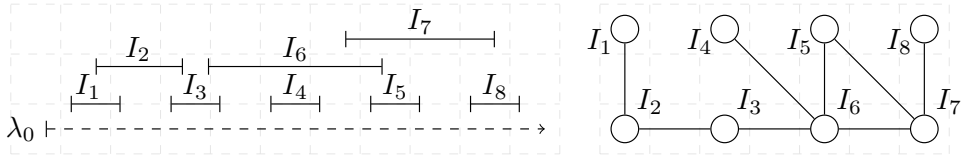
**Problem 1 (Scheduling Conflictual Deliveries Problem (SCDP)).** *Let $\delta_1, \ldots, \delta_n$ be the set of $n$ deliveries, $m$ the number of drones, and $B$ the drone's battery budget. The objective of* SCDP *is to*

*find a family* $S^* = \{S_1^*, \ldots, S_m^*\} \subseteq I$ *of* $m$ *feasible subsets with* $S_p^* \cap S_q^* = \emptyset$*, for* $1 \leqslant p \neq q \leqslant m$*, such that the overall reward* $\mathcal{P}(S)$ *is maximized. Specifically,*

$$S^* = \underset{S=\{S_1,\ldots,S_m\}\subseteq I}{\arg\max} \ \mathcal{P}(S) \qquad \textit{such that} \quad \mathcal{C}(S_i) \leqslant B \ \forall i = 1, \ldots, m$$

In the following we show the *NP*-hardness of SCDP, even for the single drone case.

An alternative way to define the set of deliveries and constraints is to observe that the set of intervals $I$ can be described as an *interval graph*. In SCDP, the set $D$ determines the set of intervals $I$ that can be pairwise compatible or in conflict with respect to their launch and rendezvous points. Given an instance of SCDP, we can visualize the intervals and their compatibility along a temporal line. Figure 2.3.2a shows the intervals corresponding to Figure 2.3.1a, in which $I_2$ is in conflict with both $I_1$ and $I_3$, whereas $I_1$ and $I_3$ are compatible.




(a) (b)

Figure 2.3.2: Delivery intervals corresponding to Fig. 2.3.1a (a); Interval graph $G$ for the example in Fig. 2.3.1a (b).

A simple way to figure out if two intervals are in conflict or not is by considering the interval graph representation. Formally, in the *undirected interval graph*, $G = (V, E)$, the vertex-set $V = I$, where each vertex uniquely corresponds to an interval $I_i \in I$, for $i = 1, \ldots, n$; and an edge $(I_i, I_j) \in E$ indicates that the deliveries $\delta_i$ and $\delta_j$ are not compatible, implying $I_i \cap I_j \neq \emptyset$. For example, in Figure 2.3.2a, $I_1$ and $I_3$ can be executed without conflicts, i.e., $(I_1, I_3) \notin E$; while $I_2$ is not compatible with both $I_1$ and $I_3$, implying $(I_2, I_1), (I_2, I_3) \in E$. Note that vertex $I_6$ has four conflicts because its degree is 4.

In the following we demonstrate that SCDP is an *NP*-hard problem by showing that the classic 0–1 Knapsack Problem (KP) is a special case of SCDP when intervals are conflict-free.

**Theorem 3.** SCDP *is NP-hard.*

*Proof.* We start proving that the simpler scenario of SCDP with a single drone is *NP*-hard. Our approach is by reduction from KP, known to be *NP*-hard [59] and defined as follows. Given a set $X = \{1, \ldots, n\}$ of $n$ items, where each item $i$ is associated with a cost $w_i$ and reward $p_i$ , and a knapsack of capacity $c$, the KP is to find a subset of $X$ that maximizes the sum of the rewards,

satisfying the capacity constraint. Given an instance of KP, we translate it as an instance of SCDP as follows: we first set the energy budget of SCDP equal to the capacity constraint of the knapsack, i.e., $B = c$. Then, we create a set of deliveries $D$ starting from the set of items $X$ as follows: for each item $i \in X$, create a delivery $\delta_i \in D$ with equal cost and reward. We create deliveries with cost $w_i$ and reward $p_i$. Finally, we create each interval $I_i = [\delta_i^L, \delta_i^R]$ with constant span $s > 0$, defining its launch and rendezvous point as $\delta_i^L = \delta_{i-1}^R + s$ and $\delta_i^R = \delta_i^L + s$, respectively, assuming that $I_1 = [0, s]$. Hence, we can observe that the deliveries are pairwise compatible in the corresponding SCDP instance. Thus, a solution for KP is a solution for SCDP and vice versa. This reduction takes polynomial time. Now, since the case with a single drone is *NP*-hard, also the general multiple drones case of SCDP is *NP*-hard. □

### 2.3.4 The Optimal Algorithm

For optimally solving SCDP in the general case with $m$ drones, we can use an ILP formulation. We enumerate the deliveries as $\mathcal{N} = \{1, \ldots, n\}$, and drones as $\mathcal{M} = \{1, \ldots, m\}$. Let $z_{ij} \in \{0, 1\}$ be a decision variable that is 1 if the delivery $j \in \mathcal{N}$ is accomplished by the drone $i \in \mathcal{M}$; otherwise it is 0. Therefore, the ILP formulation is given by:

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j z_{ij} \tag{2.1}$$

subject to:

$$\sum_{j=1}^{n} w_j z_{ij} \leqslant B, \qquad \forall i \in \mathcal{M} \tag{2.2}$$

$$\sum_{i=1}^{m} z_{ij} \leqslant 1, \qquad \forall j \in \mathcal{N} \tag{2.3}$$

$$z_{ij} + z_{ik} \leqslant 1, \qquad \forall i \in \mathcal{M}; \forall j, k \in \mathcal{N} s.t.\ I_j \cap I_k \neq \varnothing \tag{2.4}$$

$$z_{ij} \in \{0, 1\}, \qquad \forall i \in \mathcal{M}, \forall j \in \mathcal{N} \tag{2.5}$$

The objective function (2.1) aims to maximize the overall reward using a fleet of $m$ drones. About the constraints, (2.2) states that each drone has a capacity budget of $B$ in terms of energy; (2.3) forces each delivery to be executed by at most one drone; and (2.4) avoids the fact that two incompatible deliveries are performed by the same drone. (2.5) simply sets the domain of the decision variable.

In the next following, we propose algorithms for solving SCDP in scenarios involving single and multiple drones. From now on, we will interchangeably use the terms “delivery” and “interval”.

## 2.4 Algorithms for SCDP with a Single Drone

This section proposes three algorithms for SCDP with a single drone (with -S suffix) – an optimal knapsack-based algorithm (KNA-S), and two approximation algorithms, i.e., a coloring-based (COL-S) and a bin packing-based (BIN-S) whose computational time complexity is polynomial. KNA-S can be performed only when $B \in \mathbb{N}$, and it is suitable only for small values of $B$. For larger values of $B$, we devise COL-S and BIN-S. Namely, COL-S first partitions the deliveries into sets of conflict-free deliveries and then extract from them the subset of deliveries with the maximum rewards feasible with the available energy budget; whereas BIN-S first extracts the most profitable set of conflict-free deliveries and then partitions them based on the energy availability.

### 2.4.1 The KNA-S Optimal Algorithm

We propose a pseudo-polynomial time algorithm based on dynamic programming that optimally solves SCDP with a single drone, called KNA-S, requiring $\mathcal{O}(n \log n + nB)$ time and $\mathcal{O}(nB)$ space. We remark that, in this case, due to the dynamic programming approach, we require that both weights $w_i$ and budget $B$ must be integers to let the algorithm return the optimal solution.

Before proceeding, let $\sigma(i)$ be the largest index $1 \leqslant j \leqslant i-1$ such that $I_i \cap I_j = \emptyset$, and $\delta(i) = 0$ if no interval $j < i$ is disjoint from $i$. We prove the following property: if the set of intervals is sorted in non-decreasing order of the rendezvous points along the truck's route, i.e., $\delta_i^R \leqslant \delta_{i+1}^R$, then the intervals intersecting with any given interval $I_i$ are adjacent in the ordering. Our proposed dynamic programming solution exploits this property as follows: for each interval $I_i$, we can find the largest index $\sigma(i)$ that precedes $i$ without intersecting with it.

**Lemma 1.** *Let the set of intervals $I$ be sorted in the non-decreasing order of rendezvous points, i.e., $\delta_1^R \leqslant \ldots \leqslant \delta_n^R$. Given an interval $I_i$, there exists an index $\sigma(i) < i$ such that $I_k \cap I_i = \emptyset \quad \forall k \in 1, \ldots, \sigma(i)-1$ and $I_k \cap I_i \neq \emptyset \quad \forall k \in \sigma(i), \ldots, i$.*

*Proof.* (by contradiction) Assume there exist indices $\ell < \kappa < \sigma(i)$ such that the intervals between $\kappa$ and $\sigma(i)$ do not intersect with $I_i$ but $I_\ell \cap I_i \neq \emptyset$. That is, $I_k \cap I_i = \emptyset \quad \forall k \in \kappa, \ldots, \sigma(i)-1$ and $I_\ell \cap I_i \neq \emptyset$. This implies that $\delta_\ell^R > \delta_i^L$ since the intervals intersect with each other, and $\delta_k^R < \delta_i^L$ for all $k \in \kappa, \ldots, \sigma(i)-1$. So, $\delta_\ell^R > \delta_k^R$, which contradicts that $I$ is sorted in non-decreasing order of rendezvous points. In other words, $\ell < \kappa < \sigma(i)$ implies $\delta_\ell^R < \delta_\kappa^R < \delta_{\sigma(i)}^R$. □

Let us now introduce our novel KNA-S algorithm for SCDP based on dynamic programming. During the initialization phase, a table $M$ of size $n \times B$ is created and the set of intervals is sorted in non-decreasing order of rendezvous points. To compute the value in each cell of $M$, we define

a function $g(i,b)$ as the maximum reward achievable by considering the first $i$ sequences and budget $b$. Hence, we set $M[i,b] = g(i,b)$ for $0 \leqslant i \leqslant n$ and $0 \leqslant b \leqslant B$. To compute $g(i,b)$ using dynamic programming, for each set of sequences $\{0,\dots,i\}$ and for each budget $b = 0,1,\dots,B$, we first set $M[0,b] = M[i,0] = 0$ for each $i,b$. Then, for each subsequent row and column, $M[i,b]$ is the maximum between the solution $g(i-1,b)$ with an equal budget that does not consider element $i$, and the reward of element $i$ plus the optimal solution compatible with $i$. For the latter, it is necessary to find a solution that satisfies both the budget constraint and the possible intersections with the newly added element $i$. Precisely,

$$g(i,b) = \begin{cases} 0 & \text{if } i = 0 \text{ or } b = 0 \\ g(i-1,b) & \text{if } w_i > b \\ \max\{g(i-1,b), p_i + g(\sigma(i), b - w_i)\} & \text{otherwise} \end{cases}$$

Note that the above definition exploits the optimality of the sub-problems $g(i-1,b)$ and $g(i,b)$. To efficiently implement the $\sigma$ function a preprocessing phase is required. First, let us order the set of launch and rendezvous points associated with the intervals in non-decreasing order; then consider an empty stack and a pointer to the beginning of the ordered set. If the element indexed by the pointer is a rendezvous point, insert the associated interval on the stack and move the pointer by one position. Otherwise (launch point), associate the interval at the top of the stack as the predecessor $\sigma(i)$ of the interval $i$ associated with the launch point indexed by the pointer and move the pointer one position. If the stack is empty, $\sigma(i) = \varnothing$ is associated. Finally, each interval has an associated predecessor. This procedure requires $\mathcal{O}(n \log n + n)$ to sort the launch and rendezvous points and to associate the predecessors, while it takes $\mathcal{O}(n)$ space.

**Theorem 4.** KNA-S *optimally solves* SCDP *with a single drone in* $\mathcal{O}(n \log n + nB)$ *time.*

*Proof.* For the correctness of optimality, we use induction. The base case $i = 0$ is trivial since there is no interval to include in the solution. Thus, $M[0,b] = g(0,b) = 0$.

*Inductive Step*: When computing $M[i,b]$, by the induction hypothesis, $M[i-\ell_1, b-\ell_2]$ for any $1 \leqslant \ell_1 \leqslant i$ and $0 \leqslant \ell_2 \leqslant b$ are already computed correctly. There are two possible cases. If $w_i > b$, then $M[i,b] = M[i-1,b] = g(i-1,b)$ which was optimum by induction hypothesis; thus $M[i,b]$ is optimum since it is not possible to add interval $I_i$. Otherwise, if $w_i \leqslant b$ we have $M[i,b] = \max\{g(i-1,b), p_i + g(\sigma(i), b-w_i)\}$. Moreover, as before, $g(i-1,b)$ is optimum by induction. As proven in Lemma 1, the set of sequences $I_1,\dots,I_{\sigma(i)}$ contains all the sequences that do not intersect with $I_i$. Hence, by induction, $g(\sigma(i), b-w_i)$ contains the optimal solution for the set of intervals $I_1,\dots,I_{i-1}$ that is also feasible with $I_i$. The time complexity depends on the

preprocessing which is $\mathcal{O}(n \log n)$ to compute the predecessor, and on the running time $\mathcal{O}(nB)$ since the size $n \times B$ of table $M$. This permits us to calculate in constant time function $\sigma(i)$ for each interval, and hence we can fill each cell of table $M$ in constant time. □

Given that the time complexity of the optimal KNA-S algorithm is pseudo-polynomial time in the budget size $B$, in the following, we search for faster solutions that sacrifice the optimality.

### 2.4.2 The COL-S Approximation Algorithm

The COL-S approximately solves SCDP with a single drone, requiring $\mathcal{O}(n \log n + h(n))$ time and $\mathcal{O}(n)$ space, where $h(n)$ is the time required by a subroutine used in it.

The idea of COL-S is that we first adopt an optimal polynomial-time vertex graph coloring algorithm for interval graphs to divide the set of intervals into several subsets based on the minimum coloring of $I$ [60]. Each color represents a subset of deliveries without conflicts that can be assigned to a single drone. Then, we find a solution for SCDP for each subset, that is, we select the most profitable subset of deliveries in each subset of independent deliveries subject to the energy budget constraint. Finally, we return the solution with the maximum reward among all the colors. The pseudo-code of COL-S is given in Algorithm 4.

First, we find an optimal vertex-coloring for the graph induced by $I$ dividing the set of intervals $\{I_1, \ldots, I_n\}$ into $\chi$ distinct subsets. Recall that, a vertex coloring is an assignment of labels or colors to each vertex of a graph such that no edge connects two identically colored vertices. The minimum number of colors for a feasible vertex coloring in a graph $G$ is called the *chromatic number* of $G$, denoted $\chi(G)$. Moreover, a $k$-coloring of a graph $G$ is a feasible vertex coloring that assigns the integers $1, \ldots, k$ (the colors) to the vertices of $G$. Finding the minimum number $\chi$ of colors for proper coloring in arbitrary graphs is an *NP*-hard problem, but it is solvable in polynomial time for the special case of interval graphs [60].

**Algorithm 4:** The COL-S Algorithm

```
1 C_1, ..., C_χ ← coloring(I)
2 for i ∈ 1, ..., χ do
3     S_i ← max_{S∈C_i : C(S)≤B} P(S)
4 return SOL ← max_{i∈1,...,χ} P(S_i)
```

To implement `coloring` (Line 1, Algorithm 4), let us first order in a list the set of launch and rendezvous points associated with the intervals in non-decreasing order. Then consider a pointer $i$ initialized to the first point of the ordered set of launch and rendezvous points, and a min-heap

data structure that stores the colors available. The min-heap is initialized with only one color, which is the first color made available. The coloring algorithm proceeds by moving the pointer on the sorted list. If the element pointed by the pointer is a launch point, the available color in the heap with the minimum index is associated with interval $i$ that starts at the pointed launch point. If the element pointed by the pointer is a rendezvous point, the color used for the interval $i$ that finishes is inserted in the heap and made available in the min-heap for further intervals. Finally, move the pointer of one position. After extracting, if the heap is empty, insert a new color (i.e., a new available integer not used until now). Note that the overall time complexity is $O(n \log n)$ where $n$ is the number of intervals (i.e., deliveries) to be served.

The `coloring` procedure (Line 1) takes in input the set of intervals $I$, and returns as output $\chi$ subsets $C_1, \ldots, C_\chi$ of deliveries, where $\chi$ is the chromatic number of the graph[1] associated with the intervals $I$. That is, the coloring partitions $I$ into $\chi$ subsets of conflict-free deliveries, one for each color. After `coloring` is performed, for each subset $C_i$, find a subset $S_i \subseteq C_i$ with maximum reward such that $\mathcal{C}(S_i) \leqslant B$ (Line 3). Note that, subsets $S_i$ for each $i$ are feasible solutions for SCDP. Finally, we return the solution $SOL$ with the maximum reward among all the subsets $S_i$ (Line 4).

It is worth noting that finding an optimal set $S_i$ is equivalent to solving a Knapsack problem (KP) with budget $B$ on the elements $C_i$, which is an *NP*-hard problem, but which can be approximated in polynomial time.

**Theorem 5.** COL-S *provides a solution for* SCDP *for a single drone with an approximation ratio of* $\frac{\alpha}{\chi}$*, where* $\chi$ *denotes the chromatic number of the interval graph induced from the deliveries in* $I$*, and* $\alpha$ *is the approximation ratio of an algorithm that maximizes* KP.

*Proof.* Suppose we have a proper $\chi$-coloring of the set $I$ that partitions the interval sequences into $\chi$ color classes, namely $C_1, \ldots, C_\chi$. Let $S_i$ denote the set with maximum reward in color class $C_i$, (Line 3, Algorithm 1). Then, let $SOL = \max_{i \in 1,\ldots,\chi} \mathcal{P}(S_i)$ be the set with maximum reward among all sets $S_i$ such that $\mathcal{C}(S_i) \leqslant B$. Let $OPT$ be an optimal solution for the SCDP on the interval set $I$. Clearly, the intervals selected by $OPT$ are partitioned among $\chi$ colors, i.e., $OPT = \cup_{1 \leqslant i \leqslant \chi}(OPT \cap C_i)$. We first note that $\mathcal{P}(OPT \cap C_i) \leqslant \mathcal{P}(S_i)$ for $i \in 1, \ldots, \chi$. It follows directly, if either $OPT \cap C_i = S_i$ or $OPT \cap C_i = \emptyset$. Otherwise we could have $(OPT \cap C_i) \cap S_i \neq \emptyset$. In this case, $\mathcal{P}(OPT \cap C_i) > \mathcal{P}(S_i)$ will contradict that $S_i$ is the subset of $C_i$ with maximum reward since $S_i = \max_{S \subseteq C_i} \mathcal{P}(S)$.

Recall that finding an optimal set $S_i$ is equivalent to solve the KP for elements $C_i$ with budget $B$, which is an *NP*-hard problem. For each set $C_i$, let us consider two solutions $S_i$ and $S_i^*$ for the

[1] Recall that the graph has a vertex for each delivery, and an edge between any two intervals that overlap.

problem of maximizing the reward in $C_i$, where $S_i^*$ is the optimal solution to this problem and $S_i$ is an $\alpha$-approximation to $S_i^*$. Thus,

$$\mathcal{P}(SOL) = \frac{1}{\chi}\sum_{i=1}^{\chi}\mathcal{P}(SOL) \geqslant \frac{1}{\chi}\sum_{i=1}^{\chi}\mathcal{P}(S_i) \geqslant \frac{\alpha}{\chi}\sum_{i=1}^{\chi}\mathcal{P}(S_i^*) \geqslant \frac{\alpha}{\chi}\sum_{i=1}^{\chi}\mathcal{P}(OPT \cap C_i) \geqslant \frac{\alpha}{\chi}\mathcal{P}(OPT). \ \square$$

Once the optimal coloring has been computed, there are $C_1, \ldots, C_\chi$ partitions where $\chi$ denotes the minimum number of subsets without conflicts. Then, the next phase is to determine a subset $S_i$ for each $C_i$ such that the reward is maximized with the constraint of the maximum allowed budget (Line 3). In other words, we need to solve the KP on each subset $C_i$ whose size is $n_i = |S_i|$, and $\sum_{i=1}^{\chi} n_i = n$. According to Theorem 5, the COL-S algorithm can solve SCDP (with a single drone) guaranteeing a lower bound of $\frac{\alpha}{\chi}$ with respect to the optimal solution, where the parameter $\alpha$ depends on how we determine the different subsets $S_i$.

A few words to help analyze $\alpha$ and $h(n)$ in the time complexity. An approach is to rely on a fully polynomial time approximation scheme (FPTAS) with $\alpha = 1-\epsilon$, where $0 < \epsilon < 1$, requiring a computational time of $\tilde{\mathcal{O}}(n_i + (\frac{1}{\epsilon})^{2.5})$ [61], where the $\tilde{\mathcal{O}}$ notation hides poly-logarithmic factors in $n_i$ and $\frac{1}{\epsilon}$ [61]. So, a fast greedy strategy for the *fractional knapsack* problem can be exploited which guarantees $\alpha = 0.5$ [62]. Finally, to maximize $\alpha$ keeping polynomial the time complexity of the algorithm, it is possible to exploit the submodularity property[2] thus obtaining a solution that guarantees $\alpha = 1 - e^{-1} \approx 0.63$ [64] taking $\mathcal{O}(n_i^5)$ time for each $C_i$ and overall $\mathcal{O}(n^5)$ time. As a last remark we note that, in the case all the deliveries require the same cost, i.e., $w_j = \gamma \in \mathbb{N}_{\geqslant 0}$, for $1 \leqslant j \leqslant n_i$, $\alpha = 1$ because it is simply required to sort the $n_i$ elements of $C_i$ in non-decreasing order by the reward and then pick the first $\lfloor B/\gamma \rfloor$ elements. Overall this takes $\mathcal{O}(n)$ time using the selection algorithm to locate the item of rank $\lfloor B/\gamma \rfloor$ in each $C_i$ in $\mathcal{O}(n_i)$, for $1 \leqslant i \leqslant n$ [65].

### 2.4.3 The BIN-S Approximation Algorithm

The BIN-S approximately solves SCDP with a single drone, requiring $\mathcal{O}(n \log n + h(n))$ time and $\mathcal{O}(n)$ space, where $h(n)$ is the time required by a subroutine used in it.

The algorithm relies on the Bin Packing Problem (BPP). Recall that, the BPP is an optimization problem in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity, in a way that the number of used bins is minimized. Both COL-S and BIN-S search for conflict-free subsets of intervals but use different strategies: BIN-S selects a subset of independent intervals that maximize the reward and, on this, refines the intervals' selection to

[2]Submodularity [63] Given a finite set $X = \{\xi_1, \ldots, \xi_n\}$, a set function $\mathcal{F} : 2^X \to \mathbb{R}$ is submodular if for any $S \subseteq T \subseteq X$ and $\xi \in X \backslash T$, $\mathcal{F}(S \cup \xi) - \mathcal{F}(S) \geqslant \mathcal{F}(T \cup \xi) - \mathcal{F}(T)$.

satisfy the budget constraint. In more detail, we optimally compute the *maximum reward subset* of compatible intervals of $I$ in polynomial time by solving via dynamic programming the Weighted Interval Scheduling Problem (WISP). Then, we optimize the interval selection returned by WISP by grouping the solution in bins of size (energy cost) at most $B$ via a greedy implementation of the BPP. Finally, BIN-S outputs the bin with the maximum reward between all the bins. The pseudo-code of BIN-S is given in Algorithm 5.

**Algorithm 5:** The BIN-S Algorithm

1 $SOL_{\max} \leftarrow$ `weighted-interval-scheduling`$(I)$
2 $S_1, \ldots, S_\eta \leftarrow$ `bin-packing`$(SOL_{\max})$
3 **return** $SOL \leftarrow \max_{i \in 1,\ldots,\eta} \mathcal{P}(S_i)$

Initially, we find the subset with maximum reward of compatible deliveries $SOL_{\max}$ by executing the optimal algorithm for the WISP [66] on the set $I$ in input (Line 1, Algorithm 5). Then, by invoking `bin-packing` on $SOL_{\max}$, we get $\eta$ bins $S_1, \ldots, S_\eta$ such that $\mathcal{C}(S_i) \leqslant B$ (Line 2), where $\eta$ is the optimum (minimum) number of required bins. On each bin, deliveries are compatible with each other. Eventually, the bin with maximum reward, $SOL$, is returned in output (Line 3).

Note that finding the optimal division of deliveries into $S_1, \ldots, S_\eta$ bins is equivalent to solving BPP with budget $B$ on the subset of deliveries $SOL_{\max}$. However, since BPP is strongly *NP*-hard, we only consider polynomial time $\beta$-approximated solutions for it, which in turn also affects the approximation ratio of BIN-S. Recall that an approximated version of `bin-packing` returns a solution with a number of bins that is no more than $\beta \geqslant 1$ times the optimal number $\eta$ of bins.

**Theorem 6.** BIN-S *provides a solution for* SCDP *for a single drone with an approximation ratio of* $\frac{1}{\lceil \beta \eta \rceil}$*, where* $\eta$ *denotes the optimal number of bins, and* $\beta$ *is the approximation ratio of an algorithm that minimizes* BPP.

*Proof.* Let $SOL_{\max}$ be the subset of compatible intervals of $I$ with maximum reward $P_{\max}$, and $OPT$ be the optimal solution for SCDP. Suppose we have an optimal solution for BPP on $SOL_{\max}$ which partitions $I$ into $\eta$ bins $S_1, \ldots, S_\eta$ such that $\mathcal{C}(S_i) \leqslant B$. By the *pigeonhole principle* [67] there exists at least one $\hat{i}$ such that $\mathcal{P}(S_{\hat{i}}) \geqslant \frac{P_{\max}}{\eta}$. Let $S_{\hat{i}}$ be the solution $SOL_B$ of BIN-S for SCDP, and recall that $P_{\max} \geqslant OPT$ by definition since $P_{\max}$ does not consider the budget constraint. Since we rely on a $\beta$-approximation algorithm for BPP, the returned number of bins is $\lceil \beta \eta \rceil$. Hence:

$$\mathcal{P}(SOL_B) \geqslant \frac{P_{\max}}{\eta} \geqslant \frac{P_{\max}}{\lceil \beta \eta \rceil} \geqslant \frac{OPT}{\lceil \beta \eta \rceil} \implies \frac{\mathcal{P}(SOL_B)}{\mathcal{P}(OPT)} \geqslant \frac{1}{\lceil \beta \eta \rceil}. \square$$

Notice that the guaranteed approximation bound of BIN-S depends on the number of bins returned by the algorithm used to solve BPP, which in turn depends on $\beta$. Trivially, $\beta = 1$ if we

perform an optimal and computationally expensive algorithm for BPP. Another approach is to rely on the approximation algorithm devised in [68] that gives $\beta \approx 1.6$ taking $\mathcal{O}(n \log n)$ time but with a potential unbounded space complexity without any further assumption. The Best-Fit (BF) algorithm guarantees $\beta = 1.7$ in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space [69]. A faster algorithm that provides $\beta = 2$ with both $\mathcal{O}(n)$ time and space, is Next-Fit (NF) [70]. However, in this chapter, we prefer BF since it has the best trade-off.

## 2.5 Algorithms for SCDP with Multiple Drones

This section proposes three algorithms for SCDP with multiple drones (with -M suffix) – a knapsack-based approximation algorithm (KNA-M), a bin packing-based approximation algorithm (BIN-M), and a coloring-based heuristic algorithm (COL-M). Note that, these algorithms have been devised and adapted from the original single-case drone algorithms in Section 2.4.

### 2.5.1 The KNA-M Approximation Algorithm

Based on dynamic programming, the KNA-M algorithm solves SCDP with multiple drones, requiring $\mathcal{O}(m(n \log n + nB))$ time and $\mathcal{O}(nB)$ space.

The idea behind KNA-M is that we sequentially run an optimal strategy (i.e., KNA-S) on the current residual intervals for each drone. Then, we find a global solution for SCDP which comprises all the previously computed solutions. The pseudo-code of KNA-M is given in Algorithm 6.

**Algorithm 6:** The KNA-M Algorithm

```
1 Î ← I
2 for i ∈ 1,...,m do
3     S_i ← KNA-S(Î), Î ← Î\S_i
4 return SOL ← {S_1,...,S_m}
```

First, the set $\hat{I}$ characterizing the current residual intervals is initialized with all the intervals $I$ (Line 1, Algorithm 6). Then, recalling that there are $m$ drones, we iteratively (Line 2) invoke the optimal solution KNA-S on $\hat{I}$ (Line 3), which contains the intervals (deliveries) that have not been assigned to drones yet. This step makes an intermediate solution $S_i$ at each iteration for the $i^{th}$ drone. So, the current set of intervals available for the remaining drones is decreased (Line 3). Finally, the returned solution is simply the union of the previously computed sub-solutions (Line 4).

The guaranteed approximation ratio of KNA-M is proven in Theorem 7.

**Theorem 7.** *The* KNA-M *algorithm provides a solution for* SCDP *for multiple drones with an approximation ratio of* $\frac{1}{m}$*, where* $m$ *denotes the number of drones.*

*Proof.* Let $SOL = \{S_1, \ldots, S_m\}$ be the solution returned by KNA-M and let $OPT = \{O_1, \ldots, O_m\}$ be an optimal solution for the SCDP on $I$. By construction of the solution $SOL$, we have $\mathcal{P}(S_1) \geqslant \ldots \geqslant \mathcal{P}(S_m)$. Note that having $\mathcal{P}(S_j) \geqslant \mathcal{P}(S_i)$, for $i < j$, will contradict that $S_i$ is the subset with maximum reward at step $i$ of algorithm KNA-M. Moreover, since $S_1$ is the optimal solution with one drone, $\mathcal{P}(S_1) \geqslant \mathcal{P}(O_i)$, $\forall i \in \{1, \ldots, m\}$. Thus,

$$\mathcal{P}(SOL) \geqslant \mathcal{P}(S_1) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{P}(S_1) \geqslant \frac{1}{m}\sum_{i=1}^{m}\mathcal{P}(O_i) = \frac{1}{m}\mathcal{P}(OPT). \qquad \square$$

The KNA-M algorithm takes $\mathcal{O}(m(n \log n + nB))$ time, and $\mathcal{O}(nB)$ space.

### 2.5.2 The BIN-M Approximation Algorithm

Similarly to KNA-M, BIN-M iterates the BIN-S algorithm $m$ times, every time working on the remaining deliveries to find a solution that approximates SCDP. BIN-M solves SCDP with multiple drones in $\mathcal{O}(m(n \log n + h(n)))$ time and $\mathcal{O}(n)$ space. The pseudo-code of BIN-M is similar to the one already shown in Algorithm 6. The only difference resides in Line 3, where in BIN-M is iteratively invoked BIN-S. The pseudo-code of BIN-M is given in Algorithm 7.

**Algorithm 7:** The BIN-M Algorithm

```
1 Î ← I
2 for i ∈ 1, ..., m do
3     S_i ← BIN-S(Î), Î ← Î \ S_i
4 return SOL ← {S_1, ..., S_m}
```

Similarly to Theorem 7, it can be proven:

**Theorem 8.** *The* BIN-M *algorithm provides a solution for* SCDP *for multiple drones with an approximation ratio of* $\frac{1}{\lceil m\beta\eta_1 \rceil}$*, where* $m$ *denotes the number of drones,* $\eta_1$ *is the optimal number of bins, and* $\beta$ *is the approximation ratio of an algorithm that solves BPP during the first iteration.*

### 2.5.3 The COL-M Heuristic Algorithm

The heuristic algorithm COL-M solves SCDP with multiple drones in $\mathcal{O}(m(n \log n + h(n))$ time and $\mathcal{O}(n)$ space. The idea behind COL-M is that we sequentially perform a conflict-free coloring using

COL-S on the current residual intervals, so we assign a drone for each sub-partition. Then, we find a global solution for SCDP comprising the previously computed solutions. The pseudo-code of COL-M is given in Algorithm 8.

**Algorithm 8:** The COL-M Algorithm

1 $\hat{I} \leftarrow I, m' \leftarrow m, SOL \leftarrow \varnothing$
2 **while** $\hat{I} \neq \varnothing$ **do**
3 $\quad \{S_1, \ldots, S_\chi\} \leftarrow$ COL-S$(\hat{I})$
4 $\quad$ `sort`$(S_i)$ s.t. $\mathcal{P}(S_i) \geqslant \mathcal{P}(S_{i+1})$
5 $\quad SOL \leftarrow SOL \cup \{S_1, \ldots, S_{\min\{\chi, m'\}}\}$
6 $\quad \hat{I} \leftarrow \hat{I} \setminus \bigcup S_i, m' \leftarrow m' - \min\{\chi, m'\}$
7 **return** $SOL$

The set $\hat{I}$ and the number $m'$ of available drones are first initialized (Line 1). Next, until $\hat{I}$ is not empty (Line 2), an optimal coloring is performed to generate $S_1, \ldots, S_\chi$ (Line 3), which are sorted in non-decreasing order by the total reward (Line 4). Now, depending on the number of current available groups $\chi$ and drones $m'$, we assign the best $\min\{\chi, m'\}$ groups to drones (Line 5). After that, we update the current solution $SOL$ and the number of available drones (Line 6).

## 2.6 Performance Evaluation

In this section, we compare the performances, in terms of obtained reward, of the three proposed heuristics for solving SCDP with a single drone and multiple drones cases.

### 2.6.1 The Settings

In the last-mile delivery area $A$, we assume for the truck a randomly generated trip of 300km, which is a reasonable distance for a single working day [71]. The locations of the $n = \{25, 50, 75, 100\}$ deliveries are uniformly generated at random in $A$. The truck carries a total of $m = \{1, 3, 5\}$ drones that fly at a constant speed $d_s = 20$m/s [29]. We set the drone's battery $B = 5$MJ [29]. We set the drone's payload to $d_p = 5$kg [29]. With respect to these parameters, the energy cost $w_i$ for performing a delivery is computed according to the energy model presented in [29, 72], which depends on the distance to travel, the total mass of the drone plus the payload, and the drone's speed. Notice that, the payload weight affects exclusively the outbound trip since after serving the customer the drone will not carry any more a package.

About the rewards assigned to each delivery, we randomly generate them as an integer number between $[1, 100]$ according to the Zipf distribution [73] varying the $\theta$ parameter in $\{0, 0.4, 0.8, 1.0\}$. When $\theta = 0$, the rewards are uniformly distributed in $[1, 100]$; when $\theta \geqslant 0.8$, there are a few rewards with a large probability and many rewards occurring a few times. The energy costs and the delivery intervals are uniformly generated according to the Uniform distribution, assuming four *energy/delivery-interval configurations* $\Sigma_i$ (from $\Sigma_1$ with low variability to $\Sigma_4$ with high variability). In this case, the delivery interval $I_i$ of a given delivery $\delta_i$ is simply the length between $\delta_i^L$ and $\delta_i^R$ on the truck's road. With regard to the configurations, in $\Sigma_1$ the maximum energy cost is 2.5MJ and the maximum delivery interval is 1.5km, in $\Sigma_2$ this pair is 5MJ and 10km, in $\Sigma_3$ we have 7.5MJ and 20km, and finally in $\Sigma_4$ we have 30MJ and 30km. Note that, $\Sigma_1$ and $\Sigma_2$ always admit feasible deliveries, while the other two do not. Moreover, $\Sigma_4$ allows very long delivery intervals. Finally, once the delivery intervals are generated, each launch point $\delta_i^L$ is uniformly generated between $\lambda_0$ and $\lambda_{r+1}$ minus the length of the interval $I_i$.

When evaluating the performances of our algorithms presented in Sections 2.4–2.5, we compare COL-S and BIN-S with respect to the optimal solution in the single drone scenario provided by KNA-S, and we compare KNA-M, BIN-M, and COL-M with respect to the optimum solutions in the multiple drones scenario obtained from the ILP formulation. Also, as a reference for comparison (as baseline), we propose three greedy heuristic algorithms for single and multiple drone scenarios. Specifically, *Greedy Closest Rendezvous Point* (GCRP) repeatedly selects the compatible interval with the closest rendezvous point $\delta_i^R$, *Greedy Smallest Weight* (GSW) repeatedly selects the compatible interval with the smallest energy cost $w_i$, *Greedy Largest Reward* (GLR) repeatedly selects the compatible interval with the largest reward $p_i$. In the case of multiple drones, these heuristics are repeated on the residual subset of deliveries not yet assigned to the drones.

### 2.6.2 Experiment Results

Figure 2.6.1 illustrates the performances, in terms of collected rewards, of our algorithms on synthetic data. Specifically, in the first row, we present the results for SCDP with $m = 1$ (single drone), whereas in the other two rows we illustrate the results for SCDP with $m = 3$ and $m = 5$ drones, respectively. The $x$-axis reports the four different configurations *energy/interval* $\Sigma_i$, while the $y$-axis shows the ratio between the reward reported by any algorithm and that by the optimum algorithm. Clearly, the ratio is less than or equal to 1.

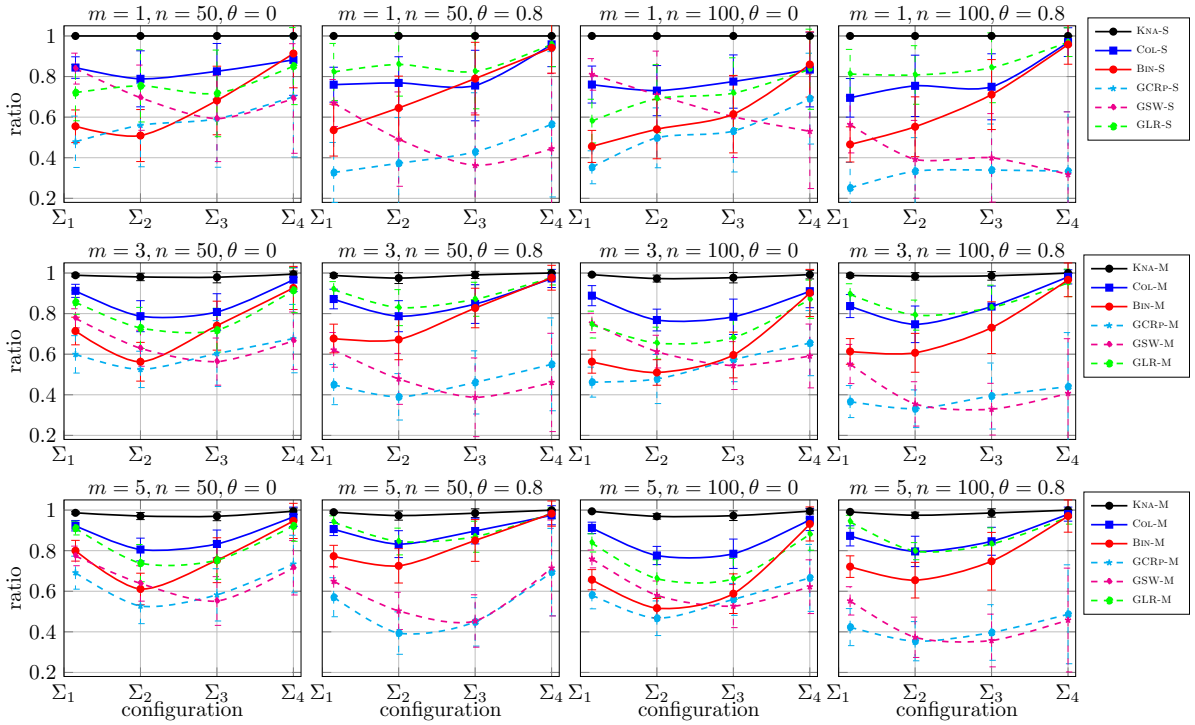



Figure 2.6.1: Performance evaluation of our algorithms. The first row compares algorithms with a single drone, while the other two rows compare with multiple drones.

### Single drone scenario

When $m = 1$, the best performing algorithms are COL-S, BIN-S, and GLR-S in general, which substantially take into account the rewards in their delivery selection rule. Instead, GSW-S and GCRP-S, whose selection rule is unlinked to the rewards, show the worst performance. This trend is particularly emphasized for the Zipf parameter $\theta = 0.8$. In this case, there are a few large-value rewards and many small-value rewards. Thus, GLR-S is facilitated in its choice. When the rewards are uniformly distributed ($\theta = 0$), COL-S guarantees better performance than GLR-S, probably because the solution considers also the compatibility of the intervals, and not only blindly the reward. In fact, while COL-S first selects the optimal sets of compatible intervals and then chooses the best one (with the largest overall reward), GLR-S could blindly select only a few large intervals, among the best ones in terms of reward, compromising the possibility to extend the goodness of the solution even if the drone has more energy to spend. Instead, in presence of unbalanced rewards ($\theta = 0.8$), GLR-S exhibits the best solution because the algorithm can select the most profitable intervals without any other consideration. The performance jump of GLR-S is high when $\theta = 0.8$. The difference between the two approximation algorithms COL-S and BIN-S is large when the variability is low (e.g., $\Sigma_1$) with COL-S that obtains $\approx$ 85% of the optimum,

while BIN-S gets $\approx$ 55% of the optimum reward. However, when the variability is higher (e.g., $\Sigma_4$), they almost perform the same (with BIN-S slightly better than COL-S). Another aspect to consider is the global loss of performance when $n$ increases. This is probably due to the increase of conflicts among the intervals, and therefore sub-optimal algorithms can suffer when selecting the deliveries for the drone without any conflict. Obviously, the ratio of KNA-S is always 1 since it is optimal.

**Multiple drones scenario**

In the multiple drone scenario, i.e., for $m = \{3, 5\}$, we observe that GCRP-M and GSW-M perform poorly, as in the single drone scenario. Accordingly, those strategies that select intervals (deliveries) taking into account either weight or rendezvous time do not perform well. The approximation algorithm KNA-M has an extremely good performance collecting $\geqslant$ 95% in any configuration. Although the guaranteed approximation ratio of KNA-M is $\frac{1}{m}$, where $m$ is the number of drones, its performance is much better than that threshold. This suggests to us that the approximation analysis for KNA-M can be probably improved. Experimentally, we have observed that KNA-M often returns the optimum, while in other cases there is only a little loss of performance, like $\approx$ 3% in average. Since KNA-M sequentially performs an optimal strategy on the residual subset of deliveries, almost always the reward obtained by the first drone is much larger than that of the second drone, and so on, while the reward that all the drones get in the optimal solution is more or less equally distributed among them (it is indeed pretty balanced). This last observation can help us find the optimal solution for the multiple drones case. Differently from the single drone case when BIN-S was employed, BIN-M performs much better in the multiple drones case when the variability is low (e.g., $\Sigma_1$). We get this behavior because most likely intervals in conflict can still be assigned to different drones, and the effect of having many intersections is less important. On the other hand, the COL-M heuristic algorithm consolidates the good performance obtained in single drone case by COL-S. In particular, we observe a quite stable trend on the four configurations which degrades on average variability configurations, i.e., $\Sigma_2$ and $\Sigma_3$. This happens probably because the coloring procedure, which computes $\chi$ feasible set of intervals, does not consider the reward to solve conflicts, therefore it may wrongly select low reward intervals at expense of more valuable intervals. However, COL-M experimentally guarantees $\approx$ 80% of the optimum in any configuration. Finally, we observe that the performances of our proposed algorithms with $m = 5$ drones are slightly better than those with $m = 3$. This is probably because the residual intervals can be assigned to other independent drones without conflicts. Notice that, we did not consider experiments with more than 5 drones because it would have been impractical to get the optimal

solution computed by the CPLEX solver.

## 2.7 Conclusion

This chapter investigated the Scheduling Conflictual Deliveries Problem (SCDP) to study the co-operation between a truck and multiple drones in a last-mile package delivery scenario. After showing that SCDP is an *NP*-hard problem, we proposed an ILP formulation that is suitable for small instances in input. We also gave an optimal pseudo-polynomial time algorithm for the single drone case and also provided additional time-efficient approximation and heuristic algorithms for the single and multiple drones cases. Finally, we evaluated the performances of the proposed algorithms on synthetic datasets. In future work, it would be interesting to investigate multi-depot multi-truck scenarios or allow drones to perform multiple deliveries at the same time or recharge their battery on charging stations. We also plan to study a more realistic dynamic environment, dealing with network communications between the depot and ground-air vehicles, and addressing possible delays and new/canceled deliveries. To address these challenges, we plan to develop online strategies to reschedule deliveries on the fly. Moreover, the SCDP assumes that the route of the truck is already pre-planned and it is given. This is however not always possible, since traffic, accidents, and road work may affect the original planned route. For this reason, a further extension of our problem might be to adjust the truck's route to optimize the expected minimum number of drones to be employed. It is worthy of investigations the package allocation problem derived from the possibility of the drone to carry more than one package at a time.

# Part II

# Investigating the Use of Drone in Smart Agriculture: the *Single-drone Orienteering Aisle-graph Problem* and *Multiple-drone Data-collection Maximization Problem*

# Chapter 3

# Drone-based Bug Detection in Orchards with Nets: A Novel Orienteering Approach

## 3.1 Introduction

The focus of this chapter is to examine the use of a drone equipped with image/video capabilities to detect pests in an orchard in a timely manner. Traditional field scouting for pest infestations is a time-consuming procedure. By utilizing drones for inspections, trees can be thoroughly and frequently examined through high-resolution imagery of all areas, including the top and bottom, tasks that would be challenging by leveraging on ground robots. With this information, combined with contextual data, early detection of pest infections can be achieved, leading to a reduction in the use of insecticides or biopesticides [74].

Figure 3.1.1: Orchards today use nets to safeguard their crops from pests and harsh weather conditions by covering the top and sides of the trees.

Orchards and vineyards are typically composed of multiple rows of trees arranged in straight lines, referred to as aisles. To protect crops from hail, many orchards have nets installed on top of the trees, as shown in Figure 3.1.1. These nets not only protect crops from adverse weather but also help reduce pest damage by making it more difficult for insects to enter and infest the trees [75,76]. However, the nets can also restrict the movement of drones, limiting them to flying only along the aisles and making the trajectories to cover the entire orchard much longer. In other words, the limited space between the treetops and the presence of the nets both pose significant risks to the drones' propellers. In order to suitably model the mobility of the drone, we propose a novel graph-based 3D data structure called *3D single-access aisle-graph*, briefly *orchard*. Such a data structure models the 3D constrained flight of a drone inside orchards wrapped with nets. The objective of the drone is to capture pictures from various observable positions of the trees modeled as vertices of the orchard. The edges of the graph represent the horizontal and vertical drone movements.

The main challenge addressed in this chapter is how to overcome the limitations of drone battery life when capturing images and videos of large orchards to detect bugs. The energy consumption of a drone does not only depend on the size of the orchard but also on the payload being carried, such as the weight of an RGB, or a multi- or hyper-spectral sensor. Also, obtaining high-quality images and videos requires the drone to fly at significantly slower speeds or even hover in place, which can lead to longer coverage times and increased energy consumption compared to regular flight.

Given these limitations, we propose of prioritizing some observable positions on trees in the orchard by assigning a deterministic "reward" value to each location, and select the drone trajectory that maximizes the overall reward under the energy limitation. The reward value associated with the locations can be determined by analyzing historical data on bug presence, weather conditions, and incorporating entomological knowledge. For instance, entomologists observed that bugs show a stronger preference for staying at the border of the orchard rather than in the middle [77], or tend to favor medium-high light exposure particularly during the early morning or late afternoon [78]. So, large reward can be appropriately assigned to the areas of the orchard where the likelihood of bug observations is the highest.

In this chapter, the objective is to determine a partial traversal of the orchard graph (i.e., an energy feasible drone's route to scout bugs) visiting the vertices (i.e., the locations) that overall return the largest reward. The Orienteering Problem (OP) can be used as a framework to model this problem by leveraging the novel graph-based 3D data structure and the concept of reward.

Our results are summarized as follows:

- We introduce a novel graph-based 3D data structure called *3D single-access aisle-graph*, briefly *orchard*, that models the mobility of drones inside orchards with nets. The vertices of the graph are weighted with a reward value which is based on the suggestions of entomologists and historical data.

- We propose an optimization problem, called *Single-drone Orienteering Aisle-graph Problem* (SOAP[1]), in which a single drone is in charge of planning an energy-feasible route inside an orchard in order to maximize the total collected reward.

- We design five algorithms to solve SOAP, i.e., a dynamic programming-based optimal algorithm (OPT), two approximation algorithms (ABP and ABA), as well as two fast heuristics (GBT+ and GBA+).

- We assess the effectiveness of our algorithms by testing them on both synthetic and real-world data, tailored to our specific problem.

The chapter is structured as follows: Section 3.2 reviews the related work. Section 3.3 formally defines SOAP. Section 3.4 presents the optimal, approximation, and heuristic algorithms for solving SOAP. Section 3.5 evaluates the goodness of our algorithms. Finally, Section 3.6 proposes conclusions and future research directions.

## 3.2 Related Work

The OP is a particular extension of the well-known Traveling Salesman Problem (TSP), in which cities have profits or rewards in addition to the cost of moving from one to another. The main goal of the TSP is to find the most cost-effective cycle that visits all the cities only once, whereas the objective of the OP is to find a cycle that visits a specific subset of the cities, to maximize the sum of profits from those visited cities, and ensure that the predetermined cost budget is not exceeded. The OP has been first introduced in [80] and proven to be NP-hard and APX-hard in [80] and [81], respectively. Many variations of the OP have been proposed in the literature, such as the rooted and unrooted versions, directed and undirected edges, planar graphs, and multi-robot scenarios. For a survey of different formulations, the reader is referred to [82]. Due to its intrinsic computational complexity, many heuristic solutions have been proposed in the literature, and exact solutions have been proposed but do not scale to problem instances with tens of thousands of vertices. A different line of research has aimed at developing approximation

[1]Insecticidal soap is used to control many plant insect pests [79].

algorithms for OP, often providing specialized results applicable only to restricted versions of the problem, such as a 4-approximation for the rooted version of the problem [83] and $(1+\varepsilon)$-approximation algorithm for the case where the graph is planar and fully connected [84]. Also this chapter provides optimal and approximation algorithms for OP in a special class of graphs.

The OP in aisle-graphs has been first studied in [85]. The authors plan a path of a robot in a vineyard to control the amount of water given to the vines and ensure the optimal moisture level. The vineyard is modeled by a 2D aisle-graphs with two accesses (one access on each endpoint of the aisle). The authors propose two greedy heuristic algorithms, GFR and GPR, to tackle this problem which allow to traverse subsets of full or partial aisles, respectively. The time complexity of GFR and GPR are $\mathcal{O}(m^2)$ and $\mathcal{O}(m^2 n)$, respectively, given that $m$ is the number of aisles and $n$ is the number of vertices (i.e., trees) in each aisle. Furthermore, the same authors in [86] study the problem of routing multiple robots, proposing three algorithms that combine GFR and GPR both in series or in parallel.

The authors in [87] improve on the results presented in [86]. They introduce an optimal algorithm called OFR-I, which improves on GFR by determining the optimal solution for the full-row policy with a time complexity of $\mathcal{O}(m \cdot \max n, \log m)$. Additionally, they propose HGC, which outperforms GPR but at the cost of a larger time complexity. Also, in two other papers, [88,89], the authors introduce a 2D aisle-graphs with only one access for each aisle and propose approximation and greedy algorithms for OP.

None of the surveyed papers address the specific aspect of 3D mobility of vehicles, such as drones, as we do in this chapter. In fact, we propose an optimal solution of OP for a variation of aisle-graphs with single access, incorporating the additional third dimension.

The OP has been also studied considering different stochastic costs to traverse the edges of the orchard [90–92]. However, due to the regularity of the actual orchards there is no substantial difference in the cost for moving from one vertex of the tree to another one, from one tree to another one, or from one aisle to the adjacent one. In any case, the movements involve repositioning over a few meters, employing a stop-and-go procedure. Hence, in our model, we have considered the energy cost for each edge constant, and hence unitary.

Finally, the authors in [93] study the OP applied to aerial 3D scanning. Although the problem may appear similar to ours since it considers the 3D scenario, their primary objective in terms of coverage differs from ours since the main goal is to cover the entire area.

## 3.3 Problem Definition

In this section, we formally present the orchard graph model, including the cost and reward functions, as well as the problem that we aim to solve.

### 3.3.1 Orchard Graph Model

A *3D single-access aisle-graph*, briefly *orchard*, and denoted by $O(m,n,l)$ is a graph data structure $G=(V,E)$ represented by the set of vertices $V$ and edges $E$ that characterize the orchard where we plan to work into. Given the orchard $O(m,n,l)$, let $m$ be the number of *rows* (aisles), let $n$ be the number of *columns*, and finally let $l$ be the number of possible *observable positions* on each tree. However, rows are all connected only via the first column (also called *backbone*) (see Figure 3.3.1) and for this reason, it is referred as 3D single-access aisle-graph. Since on each row there are $n$ trees, then the orchard has exactly $mn$ trees and $mnl$ different positions to be observed by the drone.

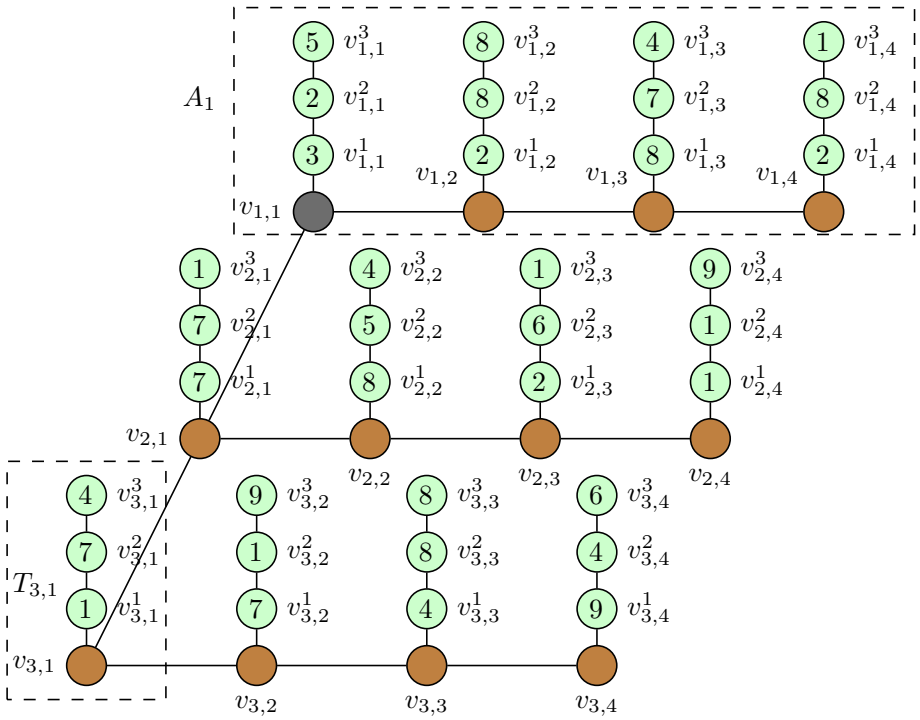



Figure 3.3.1: An orchard $O(3,4,3)$. The aisle labeled $A_1$ and the tree labeled $T_{3,1}$ are highlighted. The light-green vertices represent the rewards, while the costs have been omitted as they are assumed to be unitary. The vertex $v_{1,1}$ constitutes the depot.

The set of vertices is defined as $V = V^T \cup V^P$ where $V^T$ is the actual *set of tree roots* of the orchard and $V^P$ is the *set of observable positions* of the drone. Specifically, the former is

$V^T = \{v_{i,j} | 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n\}$ while the latter is $V^P = \{v^k_{i,j} | 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n, 1 \leqslant k \leqslant l\}$. Similarly, the set of edges is defined as $E = E^T \cup E^P$ where $E^T$ indicates *inter-tree connections*, while $E^P$ indicates *intra-tree connections*. In particular, $E^T$ is defined as follows: each vertex $v_{i,j}$ with $1 \leqslant i \leqslant m$ and $1 < j < n$ has two edges, one toward $v_{i,j-1}$ and the other toward $v_{i,j+1}$; each vertex $v_{i,1}$ with $1 < i < m$ has three edges: one toward $v_{i-1,1}$, one toward $v_{i+1,1}$, and one toward $v_{i,2}$. Therefore, the edges connected to the corner vertices $v_{1,1}$ are $v_{m,1}$ are well defined.

Concerning $E^P$, each vertex $v^k_{i,j}$ for any $i, j$ with $1 \leqslant k < l$ has two edges, one toward $v^{k-1}_{i,j}$ and the other toward $v^{k+1}_{i,j}$ forming a connected list. As per convention, let $v^0_{i,j}$ be equivalent to $v_{i,j}$. Therefore, each tree placed in $v_{i,j}$ has an edge in its associated list. In the orchard, let $r_i = v_{i,1}, \ldots, v_{i,n}$ represent the $i^{th}$ row, where $1 \leqslant i \leqslant m$, and let $c_1 = v_{1,1}, \ldots, v_{m,1}$ be the *backbone*, which is the only column connecting all rows together. Also, let $A_i$ be the $i^{th}$ *2D single-access aisle-graph*, or *aisle*, of $O$ with $n$ rows and $l$ columns, where the vertices in the columns represent the observable positions, and $1 \leqslant i \leqslant m$ (see $A_1$ in Figure 3.3.1). So, $O(m, n, l)$ consists of $m$ 2D single-access aisle-graphs, one for each aisle. Let $T_{i,j}$ be the *tree* in the orchard rooted at $v_{i,j}$ formed by the sequence of vertices $v^1_{i,j}, \ldots, v^l_{i,j}$, where $1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n$ (see $T_{3,1}$ in Figure 3.3.1). Finally, the main vertex, called *depot*, is located at $v_{1,1}$.

An example of an orchard with dimensions $O(3, 4, 3)$ is presented in Figure 3.3.1. The orchard comprises 3 rows (aisles), 4 columns (number of trees per aisle), and 3 observable positions for each tree. Therefore, there are 12 trees with a total of 36 observable positions. The main vertex $v_{1,1}$ constitutes the depot, and it is depicted in dark gray representing the initial and final location of the drone, and the root of the first tree $T_{1,1}$ of the aisle $A_1$. Moreover, the root of the trees are illustrated as the brown vertices, while the observable positions are represented by the light green vertices. It is important to note that the drone is unable to fly crossing the row of trees due to the tree's foliage, and it also cannot fly over the aisles as it is restricted by the nets (see Figure 3.1.1).

### 3.3.2 Cost and Reward Functions

In this section, we present the cost and reward functions that will be utilized to assign values (edges and vertices) to the orchard graph.

#### Cost Function

In an orchard, a drone is responsible for performing tasks such as taking pictures or videos on a specific subset of observable positions on trees. The drone is an energy-constrained flying vehicle, with a battery capacity of $B \geqslant 0$ in terms of energy. As the drone moves along edges, it expends energy from its battery. To model this energy consumption, we define the edge cost function

$\mathcal{C} : E \rightarrow \mathbb{R}^+$, which represents the energy cost of flying along any edge $e \in E$. As said, all the movements involve a repositioning of a few meters, with a stop-and-go procedure, and therefore we consider the edge cost function to be constant, i.e., $\mathcal{C}(e) = \psi$ for all $e \in E$, with a given value of $\psi \in \mathbb{N}$. In more detail, during the process of capturing pictures, the drone needs to remain stationary for a certain duration to stabilize the camera and ensure proper focus, thus consuming extra energy. Therefore, when moving between two adjacent vertices of the same tree, the additional energy required for the slightly longer hovering duration, combined with the slightly shorter repositioning time, results in energy consumption equivalent to that required when transitioning between two different trees or two different aisles. Therefore, for the purpose of simplifying the problem, the energy required for moving along edges of the graph can be assumed to be unitary regardless of the traveled distance. So, since any constant $\psi$ does not change the optimal solution, we assume unitary costs, i.e., $\psi = 1$.

**Reward Function**

Let $\mathcal{R} : V \rightarrow \mathbb{R}^+$ be the *reward function* that models the relevance of doing a specific task on vertices. The reward function provides meaningful values only for the set of observable positions $V^P$ while providing zero values for the set of trees, so $\mathcal{R}(v) = 0$ for any $v \in V^T$. According to the definition of OP, the drone can only collect the reward associated with a vertex on its first visit during a single mission (flight). Specifically, subsequent visits during the same mission to the same vertex do not yield any rewards. However, when the drone repeatedly travels on the same edge during the same mission, the energy cost is taken into account each time.

As aforementioned in Section 3.1, assigning rewards to vertices is a crucial aspect in this context and the orchard's traversal of the drone varies depending on the specific reward instance being addressed. However, the reward settlement is an aspect that falls outside our research. Our OP algorithms receive in input the rewards and return the optimal/sub-optimal traversal for the received rewards. The assigned reward values to the vertices remain constant during the drone's route planning process, and as such, there is no need to update the drone's route while it flies. If the rewards change, the path has to be recomputed from scratch. However, the rewards are not derived in real-time.

It is worth noting that in our context, larger rewards indicate areas of the orchard with a higher probability of finding bugs. Such areas are defined by leveraging the expertise of entomologists and data analysts. Different periods of the season and various hours of the day are associated with appropriate rewards. The reward decision can be adjusted based on historical data from climate or micro-climate weather stations. Therefore, studying time series data can aid in predicting bug

abundance on specific days of the year, facilitating the adjustment of reward values assigned to vertices accordingly. For example, there might be a higher occurrence of bugs in spring and late summer/early fall, and the bug abundance peaks can be determined using information collected throughout the bug population's lifecycle [94, 95]. Additionally, on those days, the sun exposure of the trees can serve as an indicator of larger rewards [96]. During the fruit growing season, the tree sites where the fruits grow would have larger rewards compared to the trunk. In the experiments in Section 3.5, since we have not collected yet enough data in the HALYomorpha halys IDentification (HALY.ID) European research project [17] to analyze time series, we will assign larger reward to the border areas of the orchard as suggested by the entomologists, to the areas where the pheromones to attract the bug are installed, or to the areas largely exposed to the sun.

### 3.3.3 Problem Formulation

In this chapter, we study the use of a drone to take photos or record videos inside an orchard to detect bugs. The collected media will then be processed to evaluate the insect abundance, but this task is not within the scope of this chapter. Given an orchard $O(m,n,l)$ with $m$ rows, $n$ columns, and $l$ observable positions in each tree, and a battery capacity $B \geqslant 0$, the goal of *Single-drone Orienteering Aisle-graph Problem* (SOAP) is to find a suitable cycle for the drone that starts and ends at the depot $v_{1,1}$ such that the collected reward is maximized and its energy cost is not larger than $B$. In fact, SOAP is a specific case of the OP problem.

The orchard data structure is basically a tree data structure (see Figure 3.3.1), and therefore cycles do not exist. In fact, given a subset of vertices $X \subseteq V$, there is a unique minimum cycle trip to/from the depot that connects all the vertices in $X$ because the structure is loosely connected. The subgraph induced by the vertices of $X$ is a tree rooted in $v_{1,1}$. For a given $X \subseteq V$, let $\overline{\mathcal{C}}(X)$ be the minimum *overall flying cost* for the drone defined as the sum of the costs on edges for the unique path for connecting all the vertices in $X$. Similarly, let $\overline{\mathcal{R}}(X)$ represent the *overall attainable reward* which is the sum of the rewards of each vertex in set $X$, as well as any other rewards in the unique cycle induced by the vertices in set $X$.

Given any subset of vertices $X$ in $O$, SOAP aims to select a subset $S$ that maximizes the reward, under the constraint that the cost of flying to connect all locations in $S$ does not exceed a given budget $B$. Formally:

$$S = \arg\max_{X \subseteq V} \{\overline{\mathcal{R}}(X) \mid \overline{\mathcal{C}}(X) \leqslant B\}, \tag{3.1}$$

where $\overline{\mathcal{R}}$ is the reward function to maximize, $\overline{\mathcal{C}}$ is the flying cost function, and $B$ is the given

budget.

Since we assume unitary costs on edges, the minimum budget $B$ in input must be $\geqslant 2$ (so, $\beta_{\min} = 2$), needed to at least visit the closest vertex from the depot, i.e., $v^1_{1,1}$, which is adjacent to it. In other words, $B = 2$ guarantees a non-trivial round trip in $O$. On the other hand, the budget $B$ is upper bounded and cannot be larger than the budget that ensures a sequential and full visit of the orchard, i.e., $\beta_{\max} = m2(n-1) + 2(m-1) + mn2l$, where the first term is the cost of traveling (back and forth) through all rows, the second term is the cost for traveling the backbone, and the third term is the cost of visiting all the trees in $O$. Any budget larger than $\beta_{\max}$ would result in a waste of energy. Hence, a meaningful value of budget is $\beta_{\min} \leqslant B \leqslant \beta_{\max}$, and asymptotically[2] it holds that $B = \mathcal{O}(mnl)$.

## 3.4 Proposed Algorithms for SOAP

In this section, we present five algorithms that are able to solve SOAP. Specifically, we devise a polynomial-time optimal algorithm, called OPT, two approximation algorithms, called *Approximation Best Point* (ABP) and *Approximation Best Aisle* (ABA), respectively, and two time-efficient greedy heuristic algorithms, called *Greedy Best Tree* (GBT+) and *Greedy Best Aisle* (GBA+), respectively.

Observe that the running time for the path computation (i.e., the execution time of the algorithms that solve our problem) do not weight on the energy of the drone because the drone itself is not responsible for computing its own route planning. In fact, this task is delegated to an external entity such as a edge/cloud service. The drone's trajectory, outputted by the OP algorithms, is then transferred and stored in the drone's memory. So, the only energy depleted by the drone is that used to load the trajectory in its local memory, and it is independent of the solver algorithm used.

### 3.4.1 The OPT Optimal Algorithm

In order to optimally solve SOAP, we propose a dynamic programming algorithm, called OPT. It exploits four tables.

For each tree $T_{i,j}$ with $1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n$, the first table $\mathbb{T}$ stores the cumulative reward that can be collected given a budget $2k$ with $0 \leqslant k \leqslant l$ from the depot in $v_{i,j}$. Obviously, $\mathbb{T}[i,j,0] = 0$ by convention. Then, we can fill the remaining cells of the table as follows: $\mathbb{T}[i,j,k] = \mathbb{T}[i,j,k-$

[2]The order of magnitude $\mathcal{O}$ not to be confused with the orchard $O$.

$1]+\mathcal{R}(v^k_{i,j})$. The OPT algorithm utilizes this table to make a decision on the number of consecutive vertices that will be included in the solution for each tree. The table $\mathbb{T}$ has a size of $mn(l+1)$.

Noting that any edge is traversed twice as there is only one path in the orchard to reach any vertex, any solution spends an even amount of budget. Without loss of generality, we calculate the optimal solution up to $\frac{B}{2}$ and we count just one unit of budget for any edge traversal.

For each aisle $A_i$, the second table $\mathbb{A}$ stores the best solution so far computed, with $1 \leqslant i \leqslant m$ and any budget $i-1 \leqslant b \leqslant \lfloor \frac{B}{2} \rfloor$. Indeed, for each aisle $A_i$, the algorithm must make a choice on which trees to include in the solution and how much budget to devote specifically to each tree. In fact, if the solution considers visiting vertices belonging to the tree $T_{i,j}$, the drone must traverse all the trees from 1 to $j-1$ in the aisle $A_i$, possibly without taking any vertex in some of those trees. So, we create a table $\mathbb{A}$ of size $m \times n \times (nl+(n-1))$ where:

$$\mathbb{A}[i,j,b] = \max_{0 \leqslant k \leqslant l} \{\mathbb{A}[i,j-1,b-k-1] + \mathbb{T}[i,j,k]\}. \tag{3.2}$$

Observe that $\mathbb{A}[i,j,b]$ considers visiting the aisle $A_i$ up to the tree $T_{i,j}$, and clearly when $k=0$ no vertices of that tree are visited.

Then, for each aisle $A_i$ with $1 \leqslant i \leqslant m$ and any budget $i-1 \leqslant b \leqslant \lfloor \frac{B}{2} \rfloor$, the third table $\mathbb{A}^*$ memorizes which is the maximum reward by varying the last tree visited:

$$\mathbb{A}^*[i,b] = \max_{1 \leqslant j \leqslant n} \{\mathbb{A}[i,j,b]\}. \tag{3.3}$$

Note that the table $\mathbb{A}^*$ has a size of $n \times (nl+(n-1))$.

Finally, the OPT algorithm has to decide which aisles to select, and for each of these, up to which tree to visit. Considering that to reach the aisle $A_i$ the drone must fly the backbone portion $c_1$ in front of the rows $1,\ldots,i-1$ up to row $i$, the optimal solution considering $A_i$ is defined by the optimal solution that includes up to the aisle $A_{i-1}$, possibly without taking any tree in some of the aisles $A_1,\ldots,A_i$. For this purpose, we create the fourth table $\mathbb{Q}$. To find the final solution, we first calculate the maximum reward with a given budget $b$ assuming that one will fly up to row $i$.

$$\mathbb{Q}[i,b] = \max_{0 \leqslant b' \leqslant \min\{b-i-1, nl+n-1\}} \{\mathbb{Q}[i-1,b-1-b'] + \mathbb{A}^*[i,b']\}. \tag{3.4}$$

Then, among $\mathbb{Q}[i,b]$, we determine the absolute maximum reward with a budget bounded by $B$ as:

$$\max_{1 \leqslant i \leqslant m, 0 \leqslant b \leqslant \lfloor \frac{B}{2} \rfloor} \{\mathbb{Q}[i,b]\}. \tag{3.5}$$

Note that the size of table $\mathbb{Q}$ is $m \times \lfloor B/2 \rfloor$.

The OPT algorithm takes $\mathcal{O}(mnlB)$ time. However, since the maximum non-trivial budget is upper bounded by $\beta_{\max} = \mathcal{O}(mnl)$, then OPT has time complexity $\mathcal{O}(m^2n^2l^2)$.

Accordingly, by the above discussion, we can state that:

**Theorem 9.** *The* OPT *algorithm optimally solves* SOAP.

*Proof.* Let us define $\mathbb{T}[i,j,k]$ as the optimal reward that can be achieved given a budget $2k$ in the tree $T_{i,j}$ starting from an initial vertex $v_{i,j}$. Let $\mathbb{A}[i,j,b]$ be the optimal reward that can be attained given a budget $b$ on only the aisle $A_i$, and must traverse the vertices $v_{i,1},\dots,v_{i,j}$, i.e., the aisle up to the tree $T_{i,j}$. Moreover, let $\mathbb{A}^*[i,b]$ be the optimal solution in the only aisle $A_i$ with budget $b$. Finally, let us define $\mathbb{Q}[i,b]$ as the optimal solution on aisles $A_1,\dots,A_i$ and budget $b$.

Let us note that we do not need to prove the optimality of tables $\mathbb{T}$ and $\mathbb{A}^*$, since the values stored in these two tables are computed through an exhaustive search. This, in the following, we will prove only the correctness (via induction) of table $\mathbb{A}$ and $\mathbb{Q}$.

Let us start with $\mathbb{A}$. For the *base case*, for any $i$, $j$, and $b = 0$ the value is 0 since no budget can be allocated to any tree. Concerning the *inductive step*, when computing $\mathbb{A}[i,j,b]$ by the induction hypothesis, we have that $\mathbb{A}[i,j-1,b-k-1]$ for any $1 \leqslant k \leqslant l$ are already computed correctly.

Since any optimal solution $\mathbb{A}[i,j,b]$ visits only the aisle $A_i$ in increasing index order, up to any tree $T_{i,j}$, and must thus traverse the vertices $v_{i,1},\dots,v_{i,j}$, $\mathbb{A}[i,j,b]$ is built starting from a sub-problem that traverses the vertices $v_{i,1},\dots,v_{i,j-1}$ and some vertices of tree $T_{i,j}$. Then, $\mathbb{A}[i,j,b]$ is based on a sub-problem that considers up to the vertex $v_{i,j-1}$ on the aisle $A_i$, leaves $(k+1)$ units of budget to reach tree $T_{i,j}$ (1 unit of budget), and can allocate budget $k$ to tree $T_{i,j}$. Recall that table $\mathbb{T}[i,j,k]$ already contains the optimal solution for tree $T_{i,j}$ on aisle $A_i$ with budget $k$. Hence, the value of $\mathbb{A}[i,j,b]$ in Eq. (3.2) is correct.

Now, assume by contradiction, that there exists a solution $\mathbb{A}'[i,j,b] > \mathbb{A}[i,j,b]$, and that is the first time that Eq. (3.2) does not provide the optimum. Let vertex $v_{i,j}$ be the vertex reached by $\mathbb{A}'[i,j,b]$ in aisle $A_i$. The solution $\mathbb{A}'[i,j,b]$ gains $\mathbb{T}[i,j,k]$ and the reward of a sub problem $\mathbb{A}[i,j-1,b-k-1]$, which is optimal. That is, $\mathbb{A}'[i,j,b]-\mathbb{T}[i,j,k] = \mathbb{A}[i,j-1,b-k-1]$, and also $\mathbb{A}[i,j,b]-\mathbb{T}[i,j,k] = \mathbb{A}[i,j-1,b-k-1]$, thus $\mathbb{A}'[i,j,b] = \mathbb{A}[i,j,b]$ contradicting the assumption that $\mathbb{A}'[i,j,b] > \mathbb{A}[i,j,b]$. Finally, since we do not know in advance the best solution inside the aisle $A_i$, the optimal solution with budget $b$ is found by computing the maximum between all $1 \leqslant j \leqslant n$, and stored in table $\mathbb{A}^*[i,b]$.

To prove the correctness of $\mathbb{Q}$ we can use similar arguments and the fact that the values in this table are based on the optimality of $\mathbb{A}^*$. □

Let us illustrate OPT using the example in Figure 3.3.1 and given a budget of 21. For convenience, we report the vertices' rewards of Figure 3.3.1 in Table 3.4.2. We compute the output of the table $\mathbb{Q}$ (see Table 3.4.3) assuming to have already computed the values $\mathbb{A}^*[i,b]$ by exploiting Eq. (3.3), for $0 \leqslant i \leqslant 2$ and $0 \leqslant b \leqslant 10$, outlined in Table 3.4.1.

Table 3.4.1: Table $\mathbb{A}^*$ considering the example in Figure 3.3.1 with $B = 21$.

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 3 | 5 | 10 | 18 | 21 | 26 | 33 | 37 | 40 | 43 |
| **2** | 0 | 7 | 14 | 15 | 22 | 27 | 31 | 32 | 35 | 39 | 40 |
| **3** | 0 | 1 | 8 | 12 | 17 | 20 | 27 | 29 | 37 | 40 | 46 |

Let us compute the best solutions when considering only the first aisle $A_1$. $\mathbb{Q}[1,1] = 3$ because in the first aisle and budget 2 (recall that the second index in $\mathbb{Q}$ is halved, so 1 means budget 2) only the vertex $v^1_{1,1}$ can be visited, which has a reward of 3. $\mathbb{Q}[1,2] = 5$ since the drone can visit both $v^1_{1,1}$ and $v^2_{1,1}$. It is worth noting that $\mathbb{Q}[1,3] = 10$ has two options, either fully visiting the first tree $T_{1,1}$ $(3+2+5=10)$ or visiting the vertices $v^1_{1,2}$ and $v^2_{1,2}$ from the second tree $(2+8=10)$. Using the same reasoning, $\mathbb{Q}[1,4] = 18$ up to the last value $\mathbb{Q}[1,10] = 43$, which is given by $v^1_{1,2}, v^2_{1,2}, v^3_{1,2}, v^1_{1,3}, v^2_{1,3}, v^2_{1,4}, v^3_{1,4}$. This means that $\mathbb{Q}[1,b] = \mathbb{A}[1,b]$, for any $b$.

Table 3.4.2: The vertices' rewards of example Figure 3.3.1. The highlighted value has aisle $i = 2$, tree $j = 3$, and observable position $k = 1$, i.e., $\mathcal{R}(v^1_{2,3}) = 2$.

| | | | | $j$ | | | |
|---|---|---|---|---|---|---|---|
| | | | | **1** | **2** | **3** | **4** |
| | | | **1** | 3 | 2 | 8 | 2 |
| | **1** | $k$ | **2** | 2 | 8 | 7 | 8 |
| | | | **3** | 5 | 8 | 4 | 1 |
| | | | **1** | 7 | 8 | 2 | 1 |
| $i$ | **2** | $k$ | **2** | 7 | 5 | 6 | 1 |
| | | | **3** | 1 | 4 | 1 | 9 |
| | | | **1** | 1 | 7 | 4 | 9 |
| | **3** | $k$ | **2** | 7 | 1 | 8 | 4 |
| | | | **3** | 4 | 9 | 8 | 6 |

Let us calculate the optimal solutions by including $A_2$ as well. From this point on, we have to reallocate the budget between the new current aisle and the previous ones. To compute $\mathbb{Q}[2,1]$, we need to evaluate the best solution when allocating a budget of 2 for $A_2$ (and 0 for $A_1$) and the best solution when providing a budget of 0 for $A_2$ (and 2 for $A_1$). In this case, $\mathbb{Q}[2,1] = \max\{0,3\}$ because with budget 2, we cannot visit any observable position in $A_2$ (this is the reason we have 0) while the best solution assigning budget 2 up to the previous aisle was 3: hence the maximum is 3. For $\mathbb{Q}[2,2]$ we have an additional comparison. In fact, we can use the budget in

different ways such as: devoting 4 only to $A_2$, spending 2 on $A_1$, or allocating 4 for $A_1$. Hence, $\mathbb{Q}[2,2] = \max\{7,3,5\}$ because in the first case, we can visit $v^1_{2,1}$ whose reward is 7, in the second case we cannot obtain any reward from $A_2$ but the best solution so far calculated up to $A_1$ with budget 2 gives 3, and in the third case 0 reward from $A_2$ and the best solution with budget 4 was with reward 5. Thus, the highest value among these is 7. With the same reasoning, $\mathbb{Q}[2,3] = \max\{14,10,3,5,10\} = 14$ up to the last value $\mathbb{Q}[2,10] = 47$ obtained assigning budget 4 to $A_2$ and budget 7 to $A_1$.

The same reasoning applies when taking into account $A_3$. In fact, for example, in $\mathbb{Q}[3,2]$ we compare the best solutions assigning budget 4 for only $A_3$, or budget 2 for $A_3$ and the best solution with budget 2 up to $A_2$, or the best solution with budget 4 up to $A_2$. Hence, $\mathbb{Q}[3,2] = \max\{0,3,7\}$ because in the first case we cannot visit any observable position in $A_3$, in the second case, we already know that with a budget of 2 the best solution gives 3, while in the third case the best solution is 7. Therefore, the highest value obtained is 7. The last value of $A_3$ is $\mathbb{Q}[3,10] = 47$.

Table 3.4.3: Table $\mathbb{Q}$ considering the example in Figure 3.3.1 with $B = 21$.

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 3 | 5 | 10 | 18 | 21 | 26 | 33 | 37 | 40 | 43 |
| **2** | 0 | 3 | 7 | 14 | 18 | 22 | 27 | 33 | 37 | 40 | **47** |
| **3** | 0 | 3 | 7 | 14 | 18 | 22 | 27 | 33 | 37 | 40 | **47** |

In conclusion, the total reward obtained is 47 with $B = 20$. As seen in Table 3.4.3, 14 units of the budget are allocated for $A_1$ and 4 units for $A_2$ as can be seen in Table 3.4.1.

### 3.4.2 The ABP Approximation Algorithm

In this section, we devise the first approximation algorithm, called *Approximation Best Point* (ABP).

The main idea behind this algorithm is that we repeatedly (and greedily) select the "most convenient" vertex among the remaining ones, i.e., the vertex whose ratio between its reward and the budget spent to reach it is the largest. The pseudo-code of ABP is given in Algorithm 9.

At the beginning, the solution $\hat{S}$ is clearly empty (Algorithm 9, Line 1). Moreover, the set of available vertices $\Omega$ to be visited is initialized with the set of observable positions $V^P$ (Line 2). Then, the main cycle is repeated while the residual budget $B$ is larger than 0 (Line 3). At each iteration of the cycle, the greedy rule selects the reachable vertex that has the largest ratio among the incremental reward, and the incremental additional cost for reaching it (Line 4). If there is such a vertex, say $v^k_{i,j}$, the drone decides to visit $v^k_{i,j}$ as well as the vertices not already visited

**Algorithm 9:** The ABP Algorithm

1 $\hat{S} \leftarrow \varnothing$
2 $\Omega \leftarrow V^P = \{v_{i,j}^k | 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n, 1 \leqslant k \leqslant l\}$
3 **while** $B > 0$ ***and*** $v_{i,j}^k \neq \varnothing$ **do**
4 $\quad v_{i,j}^k \leftarrow \arg\max_{v \in \Omega} \frac{\mathcal{R}(\hat{S} \cup v) - \mathcal{R}(\hat{S})}{\mathcal{C}(\hat{S} \cup v) - \mathcal{C}(\hat{S})}$ s.t. $B - \mathcal{C}(\hat{S} \cup v) \geqslant 0$
5 $\quad S_{i,j}^k \leftarrow \{v_{i,j}^1, \ldots, v_{i,j}^k\} \cap \Omega$
6 $\quad \hat{S} \leftarrow \hat{S} \cup S_{i,j}^k, B \leftarrow B - (\mathcal{C}(\hat{S} \cup S_{i,j}^k) - \mathcal{C}(\hat{S}))$
7 $\quad \Omega \leftarrow \Omega \backslash S_{i,j}^k$
8 **return** $\hat{S}$

on the unique and minimum length path toward $v_{i,j}^k$. Formally, the drone visits the subset $S_{i,j}^k$ of vertices in $\{v_{i,j}^1, \ldots, v_{i,j}^k\}$ that still belong to the set $\Omega$ of not visited vertices (Line 5). Then, we update the current solution $\hat{S}$, and the residual budget $B$ (Line 6). Also the set $\Omega$ is updated (Line 7). Otherwise, if there is no vertex to be visited (i.e., $v_{i,j}^k = \varnothing$), the algorithm stops cycling. Finally, the solution is returned (Line 8).

In order to derive the approximation bound, we recall the fundamental definition of submodularity:

**Definition 1** (Submodularity [63]). *Given a finite set $V = \{v_1, \ldots, v_n\}$, a set function $f : 2^V \rightarrow \mathbb{R}$ is submodular if for any $X \subseteq Y \subseteq V$ and $v \in V \backslash Y$, the marginal gain satisfies:*

$$f(X \cup v) - f(X) \geqslant f(Y \cup v) - f(Y)$$

A function $f : 2^N \rightarrow \mathbb{R}$ is monotone *modular* if the marginal gain equates the above inequality, i.e., $f(X \cup v) - f(X) = f(Y \cup v) - f(Y)$. A modular function is obviously also submodular. Precisely, it is known by [63] that an algorithm that maximizes the objective function by using a greedy rule that iteratively selects elements with the largest submodular marginal gain can achieve a $(1 - \frac{1}{e})$-approximation guarantee, where $e$ is the Euler's number. However, if the goal is to consider, in addition to a modular reward, a modular cost function as a constraint, a generalized greedy rule that iteratively selects the element with the largest ratio of the submodular marginal gain over the submodular cost achieves $\frac{1}{2}(1 - \frac{1}{e})$-approximation guarantee [97].

**Theorem 10.** *The* ABP *algorithm provides a $\frac{1}{2}(1 - \frac{1}{e})$-approximation.*

*Proof.* The result follows immediately from the approximation guarantee studied in [97]. Since both the reward function $\mathcal{R}$ and the cost function $\mathcal{C}$ in ABP are modular (namely, there is only one path between any two vertices [89]), the $\frac{1}{2}(1 - \frac{1}{e})$-approximation bound is therefore guaranteed. □

Some considerations about the time complexity of the ABP algorithm. Recalling that our orchard $O$ has $m$ rows (hence $m$ aisles), $n$ columns (i.e., $n$ trees), and $l$ observable positions for each tree, at each iteration, the vertex that has the largest ratio (Line 4) is selected among an upper bound of $mnl$ possible vertices. The calculation of the marginal reward gain and the marginal additional cost has a constant-time complexity [89], so the time complexity is $\mathcal{O}(mnl)$. Concerning the main loop (Line 3), it depends on the budget $B$, and it is executed no more than $\frac{B}{2}$ times because at each iteration one moves at least one position ahead (and we have to consider the returning path also). Accordingly, the time complexity of the ABP algorithm is $\mathcal{O}(Bmnl)$, exactly the same as the optimal OPT. Due to $\beta_{\max}$, the algorithm ABP has time complexity $\mathcal{O}(m^2n^2l^2)$. However, due to its simplicity, ABP is actually much faster than OPT, as we will see in Section 3.5.

We now illustrate how the ABP algorithm works with the example in Figure 3.3.1. We consider an instance with a given energy budget of $B = 21$. The algorithm begins to consider the ratios starting from the first $v^1_{1,1}$ which has a reward of 3 and costs 2, since the solution is empty now, and therefore the ratio is 1.5. For the second vertex $v^2_{1,1}$, its reward is 2 and costs 4, so the ratio is 0.5. This is repeated for all vertices in $\Omega$. The largest ratio is given by the vertex $v^1_{2,1}$, with reward 7, cost 4, and hence ratio 1.75. So, $v^1_{2,1}$ is added to the solution. The next vertex to be selected is $v^2_{2,1}$ with the ratio 3.5 because its reward is 7 and an additional cost of 2. In the end, the algorithm guarantees 41 as a total reward with an overall cost of 20.

### 3.4.3 The ABA Approximation Algorithm

In this section, we devise the second approximation algorithm, called *Approximation Best Aisle* (ABA), that solves SOAP. The main idea behind this algorithm is that we repeatedly (and greedily) select the "most convenient" aisle among the remaining ones. The pseudo-code of the ABA algorithm is reported in Algorithm 10.

Let $B_i = 2ln + 2(n-1) + 2(i-1)$ be the required budget for fully visiting the aisle $A_i$ from the depot [89]. The ABA algorithm assumes that $B \geqslant B_m$, i.e., the farthest aisle $A_m$ can be fully visited by the drone. This algorithm returns the best solution among two sub-solutions $\hat{S_1}$ and $\hat{S_2}$, initially empty (Algorithm 10, Line 1).

The algorithm creates the first solution $\hat{S_1}$ by selecting the aisle with the maximum reward among those that can be fully visited by the drone (Line 3). Note that by our assumption any aisle can be fully visited.

In order to build the second solution, for each individual aisle $A_1, \ldots, A_m$ we compute the optimal solution invoking the solver OPTSA proposed in [89], which solves the OP problem on any two dimensional aisle-graph consisting of a single aisle and the tree positions. These optimal

**Algorithm 10:** The ABA Algorithm

1 $\hat{S_1} \leftarrow \emptyset, \hat{S_2} \leftarrow \emptyset$
2 $\Omega = \{S_1^*, \ldots, S_m^*\} \leftarrow$ OptSa $(\{A_1, \ldots, A_m\}, B_i)$
3 $\hat{S_1} \leftarrow \arg\max_{S_i^* \in \Omega} \overline{\mathcal{R}}(S_i^*)$
4 **do**
5 $\quad S_\gamma^* \leftarrow \arg\max_{S_i^* \in \Omega} \frac{\overline{\mathcal{R}}(S_i^*)}{\overline{\mathcal{C}}(S_i^*)}$ s.t. $B - \overline{\mathcal{C}}(S_i^*) \geqslant 0$
6 $\quad \hat{S_2} \leftarrow \hat{S_2} \cup S_\gamma^*, B \leftarrow B - \overline{\mathcal{C}}(S_\gamma^*), \Omega \leftarrow \Omega \backslash S_\gamma^*$
7 **while** $B > 0$ ***and*** $\Omega \neq \emptyset$
8 **if** $B > 0$ **then**
9 $\quad \hat{S_2} \leftarrow \hat{S_2} \cup Augment\,(B)$
10 **return** $\arg\max\{\overline{\mathcal{R}}(\hat{S_1}), \overline{\mathcal{R}}(\hat{S_2})\}$

$m$ solutions $S_i^*$, with $1 \leqslant i \leqslant m$, are stored in $\Omega$ (Line 2). The time complexity of OptSa is $\mathcal{O}(B_i nl)$ where $B_i$ is upper bounded by $\mathcal{O}(nl + m)$ [89] and it is the budget required to select the entire aisle $i$, with $1 \leqslant i \leqslant m$. Then, ABA starts to create the second solution $\hat{S_2}$, so the main cycle starts depending on the residual budget $B$ (Line 4). Here, the greedy rule is to select the aisle $A_i$ that has the largest ratio "cumulative reward" (i.e., $\overline{\mathcal{R}}(S_i^*)$) to "needed budget for it" (i.e., $\overline{\mathcal{C}}(S_i^*)$) (Line 5). The selected aisle is denoted by $S_\gamma^*$. Thus, the solution $\hat{S_2}$ is augmented with $S_\gamma^*$, and the residual budget and the set of the residual pickable aisles are updated too (Line 6). The cycle stops when the residual budget is not sufficient to select another entire aisle or when the set $\Omega$ is empty. To complete the solution, any subset of positions reachable with the residual budget is suitable (Line 9). Eventually, the best solution among $\hat{S_1}$ and $\hat{S_2}$ is returned (Line 10).

**Theorem 11.** ABA *provides a solution for* SOAP *with an approximation ratio of* $\frac{1}{m}$.

*Proof.* Let $SOL$ be the set of aisles that the ABA algorithm greedily selects, and let $OPT$ be the global optimal solution of the same instance. Moreover, let $\mathcal{R}(\hat{S_1})$ be the total reward obtained by the ABA algorithm by doing the selection in Line 3, and for any $i = 1, \ldots, m$ let $\mathcal{R}(OPT_i)$ be the total reward that the optimal algorithm obtains on the single aisle $A_i$. So, we have the following:

$$\mathcal{R}(SOL) \geqslant \mathcal{R}(\hat{S_1}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{R}(\hat{S_1}) \geqslant \frac{1}{m}\sum_{i=1}^{m}\mathcal{R}(S_i^*)$$
$$\geqslant \frac{1}{m}\sum_{i=1}^{m}\mathcal{R}(OPT_i) = \frac{1}{m}\mathcal{R}(OPT).$$

Indeed, if it had been that $\mathcal{R}(S_i^*) < \mathcal{R}(OPT_i)$, then the OptSa algorithm would not have returned the local optimal solution on the aisle $A_i$. □

Some considerations about the time complexity of the ABA algorithm. Computing for $m$ times the optimal solution for each aisle $A_i$ has a time complexity of $\mathcal{O}(mB_inl)$ since the solver OPTSA takes $\mathcal{O}(B_inl)$ time for each aisle $A_i$. However, in this case $B_i$ is upper bounded by $\mathcal{O}(nl+m)$, and therefore Line 2 has a time complexity of $\mathcal{O}(mn^2l^2+m^2nl)$. The main loop in Line 4 is iterated for no more than $m$ times because the cardinality of the set $\Omega = m$, and the *maximum extraction* in Line 5 asymptotically takes $\mathcal{O}(m)$. Hence, the total time complexity of the ABA algorithm is $\mathcal{O}(mn^2l^2+m^2nl) = \mathcal{O}(mnl\max\{nl,m\})$. Whenever $B > \max\{nl,m\}$, this algorithm is faster than OPT and ABP. However, its approximation ratio is worse than that of ABP if $m \geqslant 6$.

Let us illustrate the ABA algorithm with the example in Figure 3.3.1. We take into account a budget of 21 although ABA would have expected, at least, a budget equal to 34 (i.e., able to fully visit the last aisle). The algorithm begins by considering the optimal solutions of all the aisles. For $A_1$ the total attainable reward is 58, for $A_2$ it is 52, and finally, for $A_3$ it is 68. The aisle with the most cumulative reward is $A_3$, but as previously said the drone does not have a sufficient budget for it. Thus, $\hat{S_1}$ remains empty. Since the budget is $21 < 34$, only sub-optimal solutions can be taken into account, evaluating $\hat{S_2}$. Hence, for the first aisle, the total attainable reward is 43 and costs 20, while for the second aisle the reward is 39 with costs 18 plus 2 for moving in the backbone, and finally, for the third aisle, the reward is 37 with cost 16 plus 4. Accordingly, the three ratios are in sequence 2.15, 1.95, and 1.85, respectively. Therefore, the algorithm will pick the first one. After this selection, the algorithm exits returning a solution with 43 as a total reward with an overall cost of 20.

### 3.4.4 The GBT+ Heuristic Algorithm

In this section, we propose a heuristic algorithm called *Greedy Best Tree* (GBT+), which solves SOAP in polynomial time. The algorithm involves visiting an entire aisle or discarding it. The strategy is to only consider fully visited aisles in the final solution. At each step, the aisle with the highest reward-to-cost ratio is chosen. If there is not any fully visitable tree with the residual budget, we allow to partially visit a single tree in order to avoid the wastage of budget, augmenting[3] so the solution. The algorithm finishes when $B = 0$. The pseudo-code of the GBT+ algorithm is reported in Algorithm 11.

Initially the solution $\hat{S}$ is empty (Algorithm 11, Line 1), and the main loop starts to cycle (Line 2). The drone will focus on a specific group of trees that it must visit in their entirety. This means that once the drone begins observing a position $v^1_{i,j}$ on a tree $T_{i,j}$, it must continue to the final observable position $v^l_{i,j}$. Therefore, for each tree $T_{i,j}$, the algorithm will initially calculate

[3]This is the reason behind the "+" after the algorithm's name.

**Algorithm 11:** The GBT+ Algorithm

```
1 Ŝ ← ∅
2 while B > 0 do
3     T_{i,j} ← arg max_{T_{i,j} ∉ Ŝ} R̄(T_{i,j}) / C̄(T_{i,j}) s.t. B − C̄(T_{i,j}) ⩾ 0
4     if T_{i,j} ≠ ∅ then
5         Ŝ ← Ŝ ∪ T_{i,j}, B ← B − C̄(T_{i,j})
6     else
7         Ŝ ← Ŝ ∪ Augment(B), B ← 0
8 return Ŝ
```

the total rewards and costs for visiting the entire tree, including the cost of traveling to that tree based on the current solution. In Line 3 we compute $\overline{\mathcal{R}}(T_{i,j})$ which is the sum of the rewards of vertices for the tree $T_{i,j}$, and $\overline{\mathcal{C}}(T_{i,j})$ which is the total cost for completely visiting the same tree also considering the (possible additional) traveling cost for reaching that tree given the current solution. The algorithm then selects the tree $T_{i,j}$ with the highest ratio of rewards to costs (of Line 3), and updates the current solution and budget accordingly (Line 5). This process is repeated until the budget is fully utilized. If no more trees can be chosen, the solution is improved by selecting any accessible vertex, at which point the remaining budget will be exhausted and the process ends. The final solution is then returned.

Concerning the time complexity of GBT+, recall that a tree comprises $l$ vertices. GBT+ requires to precompute the sum of rewards and costs only once, requiring $\mathcal{O}(mnl)$ in time. Given a budget $B$, the main cycle is repeated for $\mathcal{O}(\frac{B}{l})$ times. Since the number of trees in the orchard is $mn$, the time complexity of GBT+ is $\mathcal{O}(mnl + \frac{B}{l}mn)$. Hence, GBT+ has time complexity $\mathcal{O}(mn\max\{l, \frac{B}{l}\})$ which is always faster than all the previous algorithms. However, no any guarantee on the solution quality is provided.

Let us illustrate GBT+ with the example in Figure 3.3.1 when $B = 21$. We consider the rewards and costs of each tree one by one. The first tree $T_{1,1}$ has a total reward of 10 and cost of 6, resulting in a ratio of 1.67. The second tree $T_{1,2}$ has a total reward of 18 and cost of 8, and so a ratio of 2.25, and so on. When evaluating all the trees, we found that $T_{1,2}$ has the highest ratio, and therefore, it is added to the solution. After the first selection, we re-evaluate the remaining trees. For $T_{1,1}$ we have again the same ratio of 1.67. The tree $T_{1,3}$ has reward 19 and now costs 8 (ratio of 2.37), because we already considered the cost for reaching $T_{1,2}$, and so on. Now, the tree with the largest ratio is $T_{1,3}$, and it is selected. Finally, this algorithm returns 37 as the total reward and 16 as the

cost. Notice that it remains a $21 - 16 = 5$ residual budget which is not sufficient for visiting any other tree as a whole. So, in the augmenting phase the algorithm picks the "most profitable" tree among the "closest" ones (i.e., $T_{1,1}$) with respect to the current solution ($\hat{S} = \{T_{1,2}, T_{1,3}\}$). Since the residual budget is 5, on $T_{1,1}$ the algorithm can only select the first two vertices, i.e., $v^1_{1,1}$ and $v^2_{1,1}$, collecting additional reward 5. Now, the total obtained reward is 42 with a cost of 20.

### 3.4.5 The GBA+ Heuristic Algorithm

In this section, we propose another heuristic algorithm, called *Greedy Best Aisle* (GBA+), which can solve SOAP in polynomial time. The algorithm's strategy is to only consider fully visited aisles in the solution, so either visit an entire aisle or discard it. The algorithm selects the aisle with the highest ratio of rewards to costs at each step. The algorithm finishes when $B = 0$. As previously described for GBT+, we allow to partially visit a single aisle in order to avoid the wastage of budget, augmenting so the solution. The pseudo-code of GBA+ is given in Algorithm 12.

**Algorithm 12:** The GBA+ Algorithm

```
1 Ŝ ← ∅
2 while B > 0 do
3     A_i ← arg max_{A_i ∉ Ŝ} R̄(A_i)/C̄(A_i) s.t. B − C̄(A_i) ⩾ 0
4     if A_i ≠ ∅ then
5         Ŝ ← Ŝ ∪ A_i, B ← B − C̄(A_i)
6     else
7         Ŝ ← Ŝ ∪ Augment(B), B ← 0
8 return Ŝ
```

Initially, the solution $\hat{S}$ is empty (Algorithm 12, Line 1), and the main cycle starts (Line 2). The drone will start to consider only a subset of aisles visited as a whole. Basically, it follows the same reasoning as done for GBT+, with the only difference that instead of selecting full trees $T_{i,j}$ here we select full aisles $A_i$. The most profitable aisle $A_i$ in terms of ratio reward/cost, i.e., $\overline{\mathcal{R}}(A_i)/\overline{\mathcal{C}}(A_i)$, is greedily selected (Line 3). As usual, when considering the cost, we also consider the possible additional traveling cost (in the backbone) for reaching the picked aisle $A_i$. Then, we pick the best aisles until the budget is finished. It is worth noting that GBA+ can end up with a significant amount of budget, although not enough to visit an entire aisle. Thus, the algorithm can extend the solution by exhausting the budget, e.g., by selecting a portion of any reachable unexplored aisle.

Some considerations about the time complexity of the GBA+ algorithm. GBA+ requires precomputing the sum of rewards and costs only once, requiring $\mathcal{O}(mnl)$ in time. Given a budget $B$, the main cycle is repeated for $\mathcal{O}(\frac{B}{nl})$ times. Since the number of aisles in the orchard is $m$, the time complexity of GBA+ is $\mathcal{O}(mnl + \frac{B}{nl}m)$, which is bounded from above by the time complexity of GBT+.

Let us illustrate the GBA+ algorithm with the example in Figure 3.3.1. We consider $B = 21$. For the whole aisle $A_1$, the total reward is 58 with a cost of 30 and a ratio of 1.93. For $A_2$, the total reward is 52 with cost 32 and ratio 1.73, and for $A_3$, the total reward is 68 with cost 34 and ratio 2.27. So, the most profitable aisle is $A_3$. However, its cost of 34 is larger than the available budget of 21. So, GBA+ needs to augment the solution by selecting the most profitable and closest available aisle, in this case $A_1$. In the augmentation phase, the algorithm sequentially selects the first vertices of the aisle until there is budget. Specifically, the picked vertices are the ones belonging to the trees $T_{1,1}$ and $T_{1,2}$ plus the individual vertices $v^1_{1,3}$ and $v^2_{1,3}$. Such a visit has a cost of 20, and the total reward is 43.

In Table 3.4.4, we summarize the running time and the performance of the presented algorithms. Note that all the time complexities are polynomial because a budget $\beta_{\max} = \Theta(mnl)$ is sufficient to traverse the entire orchard.

Table 3.4.4: Comparison between the algorithms that solve SOAP.

| Algorithm | Time Complexity | Approx. ratio |
|---|---|---|
| OPT | $\mathcal{O}(Bmln)$ | 1 |
| ABP | $\mathcal{O}(Bmln)$ | $\frac{1}{2}(1-\frac{1}{e})$ |
| ABA | $\mathcal{O}(mn^2l^2 + m^2nl)$ | $\frac{1}{m}$ |
| GBT+ | $\mathcal{O}(mnl + \frac{B}{l}mn)$ | – |
| GBA+ | $\mathcal{O}(mnl + \frac{B}{nl}m)$ | – |

## 3.5 Performance Evaluation

Our algorithms have been implemented in C++14 and run on an Intel i7-10genK computer with 16GB of RAM[4]. We assessed the performance of five algorithms for solving SOAP, i.e., OPT, ABP, ABA, GBT+, and GBA+, by evaluating the total rewards obtained with respect to the optimal one.

[4]The code will be available on GitHub here: https://github.com/TheAnswer96/TMC

We also evaluated the experimental running time. Note that when we refer to "running time", we mean the time required by the computer to compute the drone trajectory, and we do not refer to the drone flying time which is related to the budget $B$ given in input to the problem. Indeed, the chapter contributes to understanding the complexity of finding the drone trajectory and its impact on the collected reward. While the drone trajectory is currently computed offline by an external service, one can envision a future scenario where such a service needs to be provided to a community of farmers. Consequently, it is crucial to minimize the computational effort as much as possible, while ensuring that the collected reward, which will guide farmers in their decisions on orchard interventions, remains uncompromised. Furthermore, in countryside regions dedicated to intensive orchards, the values of $m$ and $n$ can become very large. Therefore, it is worthwhile to study lightweight optimization solvers that offer a tradeoff between the running time and the collected reward performance.

We conducted tests on three distinct reward workloads, labeled $W_1$, $W_2$, and $W_3$:

- $W_1$: large data set of rewards randomly generated;
- $W_2$: small data set of rewards referring to monitoring insects collected in a case-study orchard, and
- $W_3$: large data set of rewards referring to spraying management in smart agriculture.

We ran each algorithm in various scenarios, varying the number of rows $m$, the number of columns $n$, the number of observable locations $l$, and the distribution of rewards by adjusting appropriate parameters. For each scenario, the algorithms have been tested varying the drone's budget $B = 5\%, 10\%, 15\%, 20\%, 40\%, 60\%, 80\%$ (plotted on the $x$-axis) as a percentage of a given maximum $\beta_{\max}$ budget. On the $y$-axis, we plot the ratio $\rho$ of the total reward collected by the tested algorithm to the total reward collected by OPT.

For $W_1$, for each value of $B$, we plot the average of the results from 33 workload instances along with their 95% confidence interval on the $y$-axis. To evaluate the workload $W_2$, we create the data set of rewards using prior knowledge of the orchard and the position of the pheromones. Lastly, for $W_3$, we use 10 reward instances from a spraying application for smart agriculture that has been adapted for SOAP. In principle, $W_1$ is created to evaluate the robustness of the algorithms using random rewards based on a specific random distribution. On the other hand, $W_2$ and $W_3$ are constructed to emulate a real-world scenario where the rewards of vertices are determined based on bug monitoring and spraying management, respectively.

### 3.5.1 With Synthetic Data set

In our experimental workload $W_1$, we utilized the ZipF distribution [73] to randomly generate rewards on vertices by varying the $\theta$ parameter. This allowed us to study the impact of reward variability on our algorithms. Our evaluation assumed rewards in the range of $[0, 100)$ and varied $\theta$ as $\{0, 0.8, 2\}$. When $\theta = 0$, the rewards are distributed uniformly across all values. As the value of $\theta$ increases, the distribution of rewards shifts toward smaller rewards being more frequent than larger ones. In particular, $\theta = 2$ represents a very high variability with a very few items having large rewards and a plethora of small rewards. To vary the layout of the orchard, we set the number of rows as $n = \{50, 25\}$, the number of columns as $m = \{25, 50\}$, and the number of observable locations as $l = \{3, 5\}$. Note that these three distributions of rewards may correspond to three moments of the season. $\theta = 2$ indicates the early days of the season when the overwintered bugs emerge and require exposure to the sun to find energy. During this time, the bugs are primarily found on the treetops, exposed to the sun. $\theta = 0.8$ corresponds to the end of the season when the bugs are attached to the mature fruits on the trees, and $\theta = 0$ represents spring, where no pests have emerged yet, and any spot could potentially hide one. The results of the evaluation of synthetic data sets can be seen in Figure 3.5.1.

#### Impact of $\theta$

To start, we examine the effect of $\theta$. As we can see, the plots to the left have $\theta = 0$, the ones to the center have $\theta = 0.8$, while the others to the right have $\theta = 2$. From the left to the right, the variability of the reward increases. Although the two greedy algorithms GBT+ and GBA+ work almost the same when $\theta = 0$, whose gap is about 10% in favor of GBT+, their behavior is much different when $\theta$ increases. When $\theta = 2$, the GBA+ algorithm tends to select large areas of the orchard that have many low reward vertices, resulting in the budget being wasted due to the high variability of the reward and lack of adherence to any locality rule. Despite the GBT+ algorithm having the same logic as GBA+, it is much more conservative than GBA+ because in its selection rule it considers much smaller portions of the orchard, and therefore the chance of wasting energy is more limited. In fact, one can observe that the gap between the two heuristics, with $\theta = 2$, is pretty large and it is up to 40% with a low budget. A similar general behavior can be observed with the two approximation algorithms ABP and ABA. Indeed, ABA works similarly to GBA+, while ABP is much more accurate than GBT+ in its greedy selection.

The ABP algorithm is particularly efficient when the variability is high with $\theta = 2$, whose performance is close to the optimal one for larger values of $B$. It is important to point out that in Figure 3.5.1 we plot the ratios of the total collected reward among the compared algorithm and

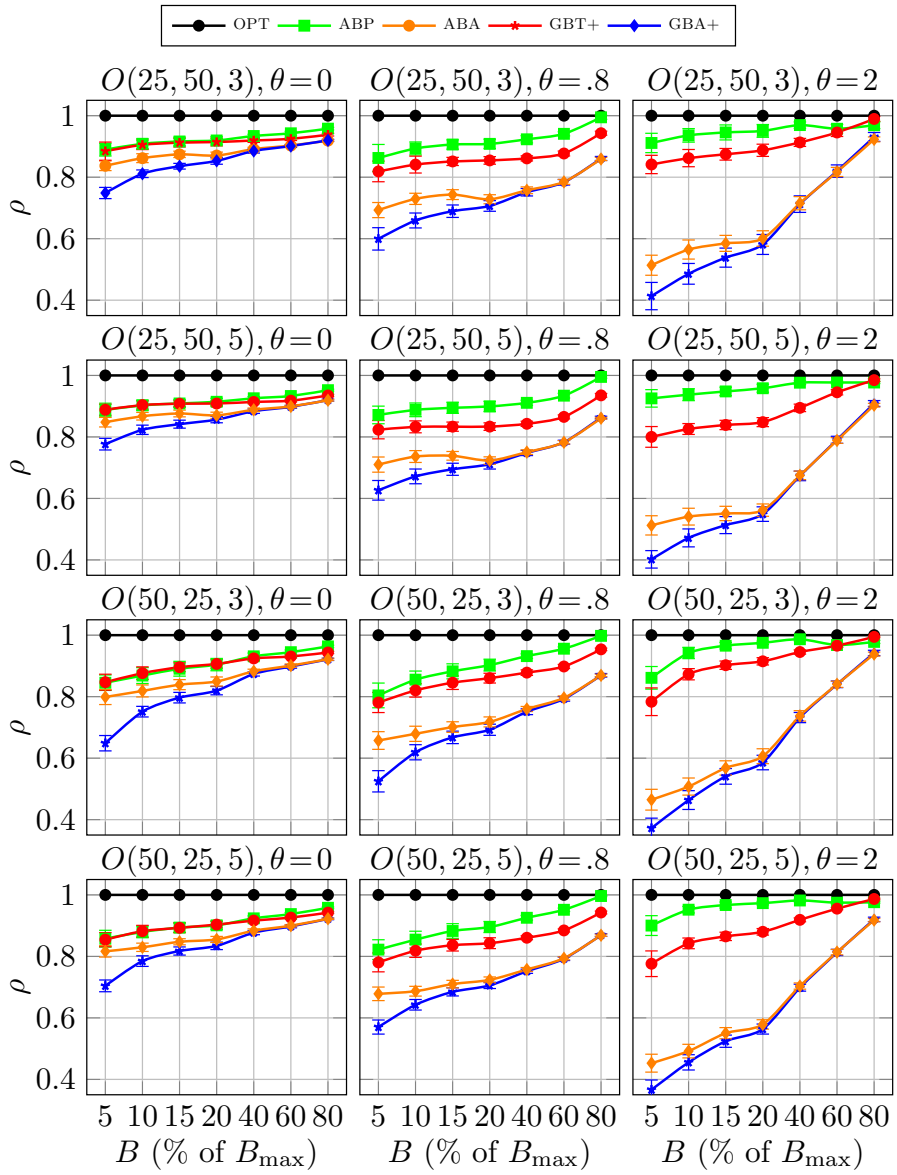



Figure 3.5.1: $W_1$ (performance) with randomly generated synthetic data sets.

the optimal one. It could be counter-intuitive to see that an algorithm has a "drop" in performance when the budget increases (e.g., ABP from $B = 40\%$ to $B = 60\%$ of $\beta_{\max}$). Obviously, the $\overline{\mathcal{R}}$ function to maximize is monotone non-decreasing. However, this does not guarantee that also the $\rho$ function is monotone non-decreasing.

### Best performing algorithms

In general, among the approximation algorithms and the heuristics, the best-performing algorithm in term of collected reward is ABP. Indeed, when the assigned budget is very large (say, 80%), often it reaches the optimal solution. However, when the budget is small (say, $\leqslant 40\%$), sometimes

GBT+ matches ABP. This is particularly emphasized when $\theta = 0.8$, where the gap between the two is almost negligible. Although GBT+ well performs with a low budget, its performance slowly degrades when the available budget increases. This is primarily because GBT+ is essentially a greedy strategy and it may make poor choices resulting in most of the budget being wasted.

Comparing the two heuristics GBT+ and GBA+, we can see that in general GBT+ obtains better performance than GBA+. However, when the budget is very large (say, 80%), the two heuristics almost work the same. When the budget is limited, GBT+ can choose the most beneficial tree based on the current budget, while GBA+ is more constrained by the budget and is forced to select a nearby aisle regardless of its reward.

In Figure 3.5.1 we can also observe that ABA is no worse than GBA+. This is due to the fact that the approximation algorithm ABA knows the sub-optimal solutions of each aisle for any value of budget. Finally, the two approximation algorithms ABP and ABA show a big gap when the budget is very small (say, $\leqslant 10\%$), while their performance almost converge when the budget is large (say, 80%).

Concerning the guaranteed approximation bounds, both are largely beyond their aforementioned thresholds. In particular, ABP always guarantees a ratio of at least $0.9 \geqslant 0.32 \approx \frac{1}{2}(1 - \frac{1}{e})$.

### Impact of the tree length

By increasing the number $l$ of observable positions, we see that the performance of the algorithms slightly differs. In particular, the performance of GBT+ in terms of $\rho$ partially decreases. This is mainly because the drone has to visit a tree much deeper, which results in possibly unnecessary energy consumption. On the other hand, the ABP algorithm is much more resilient in this aspect because it would only select the most appropriate observable position on such trees, instead of visiting them completely. Counter-intuitively, GBA+ improves its performance when $l$ increases. Probably, it is more convenient to visit a longer aisle completely than to waste energy traveling toward many other deeper trees on different aisles.

### Impact of the orchard size

We also compare the performance of two different layouts: one with more columns than rows ($m < n$) and another with more rows than columns ($m > n$). The results indicate that in the former case ($m = 25, n = 50$) the performance is generally slightly better than in the latter case ($\approx 5\%$). This difference in performance is probably influenced by the energy required for the drone's backbone flight.

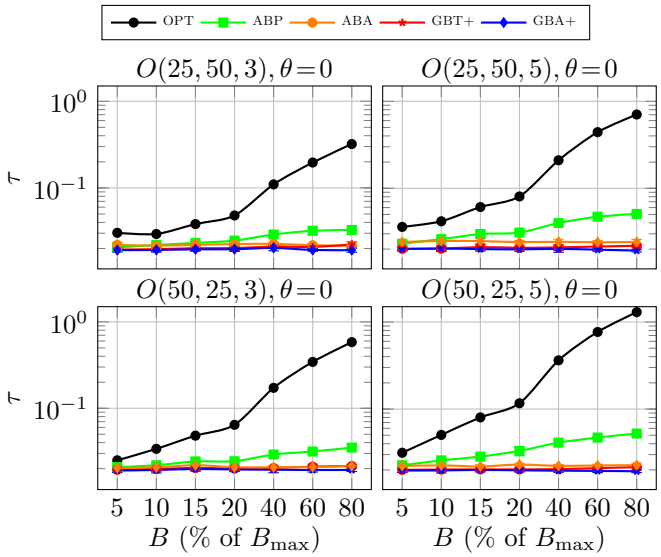



Figure 3.5.2: $W_1$ (running time in seconds) with synthetic data sets.

**Running time**

In Figure 3.5.2 we plot the experimental running time $\tau$ in the $y$-axis (in seconds, represented in logarithmic scale), varying the orchard parameters $n$, $m$, and $l$ while keeping fixed $\theta = 0$. The first important observation here is that the heaviest algorithm, as expected, is the optimal one (OPT), with computations up to 1.5s. As noticed in Section 3.4, ABP is much lighter than OPT. Although the two algorithms have the same asymptotical time complexity $\mathcal{O}(Bmnl)$, ABP is $\approx$ 15-20 times faster than OPT because it just repeats the selection of the maximum in a large set. The ABA algorithm is even 2 times lighter than ABP, however, its performance in terms of reward has a drop of $\approx$ 40% on the same instances. Finally, GBA+ is the fastest in absolute, but it is much more profitable to perform GBT+ due to its good trade-off performance/running time.

To conclude, when $m$ and $n$ are fixed and $l$ increases, the experimental running time $\tau$ obviously increases. However, fixing $l$ and swapping $m$ and $n$, the running time is more when there are more rows than columns.

### 3.5.2 With a Real-world Scenario

To evaluate the performance of our algorithms in a real-world setting, we create a workload called $W_2$. This use-case experiment is in the context of the HALY.ID project [17] that aims to scout the brown marmorated stink bug (BMSB), i.e., the *Halyomorpha halys* bug. The orchard used for the use-case, located in Carpi, Modena, Italy, has 12 rows and 15 columns, with a total of 180 pear trees, and we set 3 observable positions per tree, resulting in an $O(12,15,3)$ configuration.

In our small-scale preliminary use-case [9], our goal is to detect the presence of the BMSB by leveraging two different strategies. In this workload, we concentrate the rewards in (i) a few trees where the pheromones are located, (ii) at the border of the orchard, (iii) in the East side of the orchard, and (iv) on the West side of the orchard following the insights provided by entomologists. The results are reported in Figure 3.5.3.

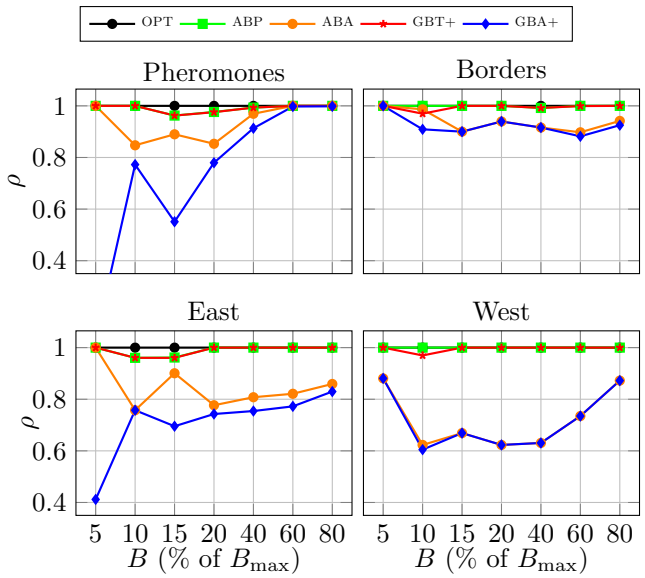



Figure 3.5.3: Performance of $W_2$.

Figure 3.5.3 (top-left) depicts the results when only a small number of trees have been treated with pheromones. We set three observable positions on each tree: bottom, middle, and top. We redistributed the rewards in this way to encourage the drone to prioritize the tops of the trees, as this is where pests are most likely to be found according to entomological knowledge. To be specific, we redistributed the rewards on each tree $v_{i,j}$ by allocating 20%, 30%, and 50% of the reward to $v^1_{i,j}$, $v^2_{i,j}$, and $v^3_{i,j}$, respectively, by utilizing pheromones.

The more penalized algorithm is GBA+ which is forced to follow an aisle even if the pheromones have been located in only one tree of that aisle. The selection of the best aisle alleviates the bad performance result for ABA.

The three plots of Figure 3.5.3 (top-right and bottom) report the results when the rewards are distributed as suggested by the entomologists experience. In the "Borders" case, the reward is assigned with several rays of decreasing relevance, so that the trees that fall on the perimeter of the orchard will have larger reward values, while the more internal trees will receive less reward accordingly. Notice that once again the larger values of reward are assigned to the highest observable positions of the trees. In this case, all the algorithms reach more than 80% of OPT with a significant stable trend through the diverse budgets. Hence, the choice of the algorithm is

not decisive in determining how to intervene in the orchard.

Then, in the "East" (resp., "West") scenario larger values of reward are distributed, assuming left as the East, from the left (resp., right) to the right (resp., left) respecting the distribution of the bugs in the morning (resp., afternoon). This paradigm is not artificial since orchards are built in order to provide an adequate level of sunlight from morning to afternoon. Therefore, focusing on the East scenario, the trees that lay on the left part of the orchard have larger values, with their values that smoothly diminish moving towards the right border. Also in these two cases, the highest observable positions will have more reward than the lowest ones. It is evident that the algorithms that select entire aisles are penalized with respect to the others. On the other hand, ABP and GBT+ once again confirm their notable results. Hence, in this case, as in the Pheromones case, it is always preferable a tree-wise greedy selection instead of an aisle-wise greedy selection.

### 3.5.3 With Real Data set

In the workload $W_3$, we employed a real data set obtained from a different smart agriculture application [89] to evaluate our algorithms on larger orchards. The field is made up of $137 \times 107$ trees. The data set, as it was originally, had only one reward for each tree which represented the water needed by the tree. To make it suitable for our spraying application, we divided each tree into three observable positions: bottom, middle, and top, and we redistributed the rewards accordingly. Therefore, the resulting orchard is $O(137, 107, 3)$. A unique characteristic of the workload $W_3$ is that the requirement for pesticide (i.e., reward) is divided into macro areas that encompass several trees. As before, the rewards change gradually on each tree $v_{i,j}$ by putting 20%, 30%, and 50% of the reward to $v^1_{i,j}$, $v^2_{i,j}$, and $v^3_{i,j}$, respectively,

Figure 3.5.4 (left) shows the results in terms of $\rho$. Differently from what happened before for all other scenarios, in $W_3$ the GBT+ algorithm performs very well with $B \leqslant 40\%\beta_{\max}$, outperforming the approximation algorithm ABP. This is probably because the reward in such areas smoothly increases or decreases, and therefore the drone has the convenience to visit trees completely, as GBT+ does. The opposite trend can be observed with the two algorithms that have the "aisle" as a greedy selection rule. As usual, ABA always outperforms (or equals) the greedy GBA+.

Finally, Figure 3.5.4 (right) depicts the results, in terms of $\tau$, of the proposed algorithms. Here, since the size of the orchard is much larger, the OPT algorithm returns the optimal solution in $\approx$ 1s with a very low budget ($B = 5\%$), while requiring $\approx$ 100s with a very large budget ($B = 80\%$). On these larger instances, we can observe the goodness of the ABP algorithm, which obtains very good solutions requiring about a $100^{th}$ of the running time of the optimal solution. The two heuristics GBT+ and GBA+ return both a solution within a fraction of a second and

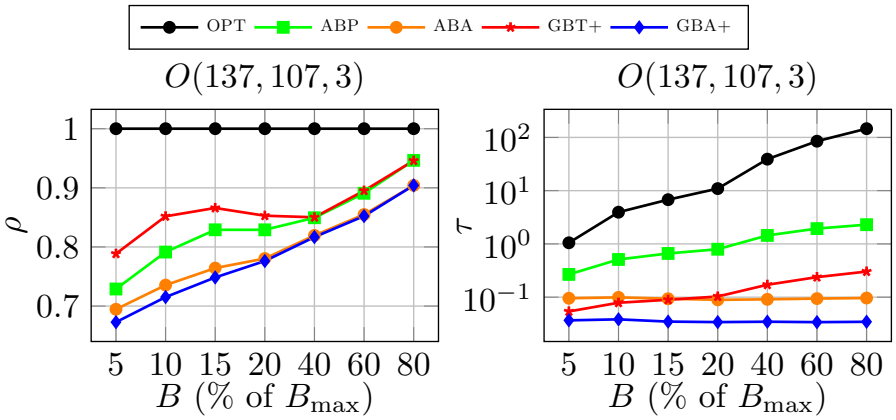



Figure 3.5.4: Performance (left) and running time (right) of $W_3$.

therefore they are particularly suitable when the size of the orchard starts to be quite large (and hence heavy) for GBT+ or ABP.

## 3.6 Conclusion

In this chapter we introduced a new problem, SOAP, and a 3D graph model that represents the drone movements in really constrained orchards. We first proposed a polynomial-time algorithm to optimally solve SOAP and then two approximation algorithms along with other two heuristic algorithms with a much reduced computational time cost. We analyze both theoretically and via numerical evaluation the proposed algorithms using both synthetic and re-interpreted real data sets. In general, we found that the solutions that make choices based on tree granularity are better than those that work at aisle granularity in all scenarios, but especially in the real-world scenario. In particular, the greedy algorithm GBT+ is light and performs well. However, when the size of the orchards is large, OPT has definitely much better performance. For large orchards, the approximated ABP solution conjugates good reward- and light time-performance.

Further research could include studying the online SOAP for which the reward values are discovered when the drone flies and hence the drone route has to be recomputed online. Also symbiotic systems involving multiple drones and ground devices, including battery recharging stations for drones, might be of future interest.

# Chapter 4

# Wireless IoT Sensors Data Collection Reward Maximization by Leveraging Multiple Energy- and Storage-Constrained UAVs

## 4.1 Introduction

In this chapter, we consider that a set of multiple homogeneous drones is responsible for collecting data from ground IoT sensors. An example of a data collection scenario using two drones is depicted in Figure 4.1.1. We assume that a set of heterogeneous ground sensors is randomly deployed on an area for sensing particular phenomena. Due to the fact that the deployment area can be very large, the sensors cannot directly transfer their perceived data to the depot. In our proposed architecture, drones flying over the area are responsible for performing a mission (a route) to/from the depot, with the objective of selectively collecting the data from the sensors. However, the drones themselves are limited in terms of energy battery (when flying and hovering) and available memory storage (when collecting data). In principle, due to the two limitations mentioned above, drones cannot collect all the data from all deployed sensors, but they have to plan a proper route and use parsimoniously both their available energy and storage. Moreover, in harsh environments, the drones could not immediately transmit the collected data to the cloud/depot because the Internet connectivity can be absent, and hence they need to keep the data to their storage, which is limited in capacity, until they finally reach the main depot.

In our proposed context, certain sensors' data are more critical and should be collected with

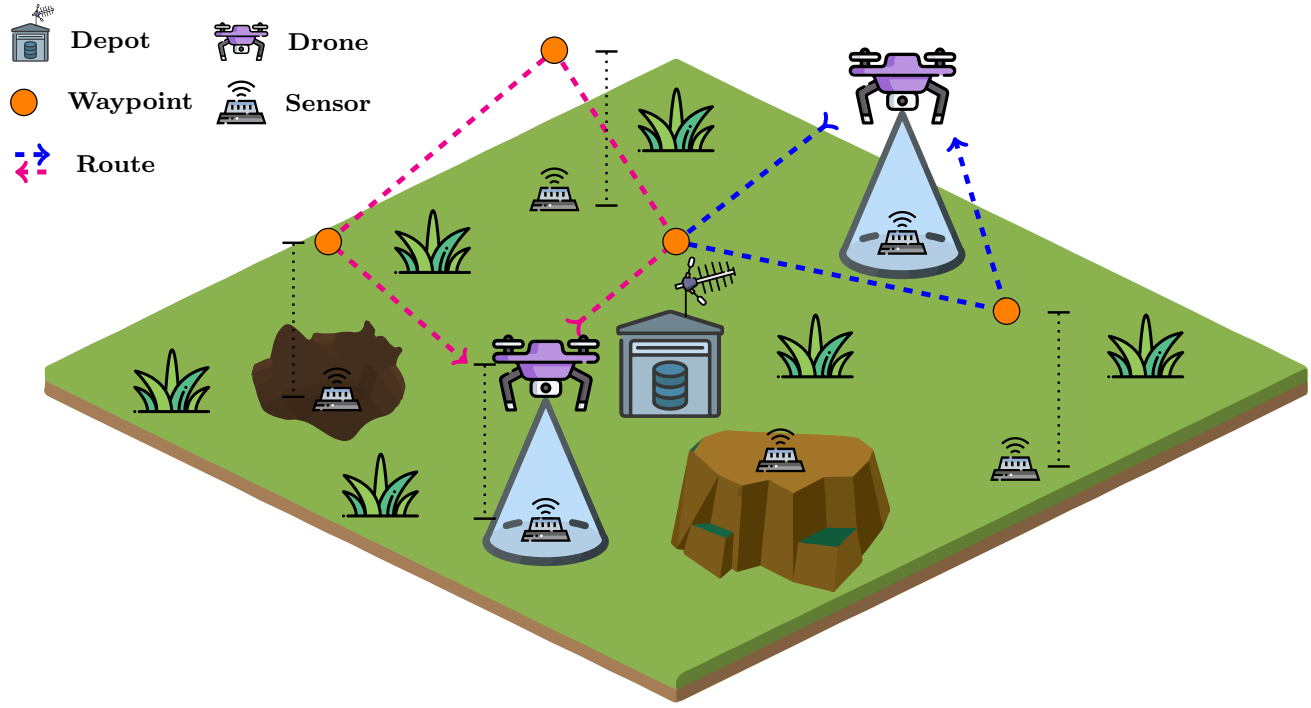



Figure 4.1.1: The sketched representation of our application. The surface is not flat, and therefore the sensors have different heights with respect to the depot.

higher priority than others. This implies that some data are more relevant and important than others. Moreover, since some older data can be lost if not offloaded in a timely manner, it is crucial to consider data more relevant if stored in a sensor that has less space left. Therefore, the farmer of the field to be monitored must prioritize these sensors over others, based on the criticality of the data and the likelihood of data loss. To model the relevance and consequent prioritization of data to be collected by drones, a specific reward is assigned to each sensor data based on the data relevance. Urgent data requiring immediate analysis is given a higher reward than regular data, and data that may be lost due to the shortage of available local storage is also prioritized.

The primary goal pursued in this chapter is maximizing the total reward obtained by collecting the most relevant data using a fleet of drones. However, this must be achieved while ensuring that the energy cost of each drone's mission does not exceed the battery budget, and that the total collected data do not exceed the storage limit on each drone. To the best of our knowledge, this is the first time that a fleet of homogeneous drones is in charge of collecting data from a set of heterogeneous ground sensors while simultaneously taking into account both the available energy and storage for the drones.

The contributions of this chapter are summarized below.

- We define a novel optimization problem, called *Multiple-drone Data-collection Maximization*

*Problem* (MDMP), whose goal is to collect the most relevant data by leveraging drones, i.e., maximizing the total obtained reward, while ensuring that each drone's mission energy cost does not exceed the battery budget, and the total collected data do not exceed the storage limit on each drone;

- We formally prove that MDMP is *NP*-hard (even for the single-drone scenario) because the Team Orienteering Problem (TOP) can be seen as a particular instance of MDMP;
- We devise an Integer Linear Programming (ILP) formulation for optimally solving MDMP, only suitable for small-sized inputs;
- We propose approximation and heuristic algorithms for obtaining suboptimal solutions for the single- and multiple-drone scenarios for inputs of any size;
- We thoroughly evaluate the performance in terms of collected reward of our algorithms on randomly generated synthetic data.

The rest of the chapter is organized as follows. Section 4.2 reviews the related work. Section 4.3 formally defines MDMP showing its *NP*-hardness, and proposes the ILP-based optimal algorithm for it. Section 4.4 and Section 4.5 present suboptimal approximation and heuristic algorithms for solving MDMP for the scenario of a single and multiple drones, respectively. Section 4.6 evaluates the effectiveness of our algorithms on randomly generated synthetic data, and Section 4.7 offers conclusions and future research directions.

## 4.2 Related Work

Many papers have been proposed in the realm of data collection in sensor networks with the help of drones.

In [98, 99], the authors consider the problem of scheduling the flight of a drone in charge to maximize the utility due to the data collected in a sensor network composed by homogeneous sensors deployed on a flat surface. The drone has the ability to simultaneously collect data from multiple sensors and its hovering time depends on the size of data to be collected. The authors discretize the possible hovering points for the drone in order to limit the number of them. A similar scenario is also considered in [100], in which the objective is to find the tour that maximizes the utility of the collected data considering the data transfer divided into time slots of equal width. Furthermore, the paper in [101] considers the problem of determining the minimum number of UAVs to be deployed to collect all the data from sensors on a flat area without exceeding a given budget time. Two algorithms are proposed to solve the problem. In contrast to the three previously mentioned works, we consider several crucial factors that have been not accounted for,

namely the presence of heterogeneous sensors at different elevations, the energy consumption required for flight, the drone's storage capacity limitation, as well as the presence of a fleet of multiple drones. Furthermore, to prevent potential bandwidth saturation, we do not permit the simultaneous collection of data from multiple sensors. These considerations are of significant importance when designing a drone-based data collection system, and in this chapter we address these challenges in a comprehensive manner.

The problem of scheduling the UAV's tour which minimizes the maximum energy consumption for all the sensors is studied in [102]. The authors jointly consider the sensors' wake-up and the UAV's path by formulating a mixed-integer non-convex optimization program, and a suboptimal algorithm which iteratively applies a successive convex optimization technique. The authors in [103] propose a clustering algorithm and meta-heuristics to address a similar problem where the sensors are deployed in a hilly terrain, and the single UAV is not energy-constrained. In contrast to the above papers, which consider homogeneous sensors relying on continuous communications or even not energy-constrained drones, we propose approximation and time-efficient heuristic solutions for a system with multiple drones and heterogeneous sensors. Additionally, all the previous works only offer computationally expensive exact solutions or meta-heuristic solutions, whereas we also propose more time-efficient deterministic methods.

The problem of maximizing the *freshness* of the data collected by a UAV has been studied in [104]. In particular, two problems of Age-of-Information (AoI) data collection are formulated to minimize both the sensors' maximal and average AoI. In [105], the authors propose a framework for controlling the flight speed of the UAV to improve the *fairness* of data collection. They formalize fairness as a metric that depends on the energy level of the sensor nodes and on the amount of data to be sent to the UAV. Specifically, since only cluster heads have to transfer data collected via multi-hop from the other nodes to the UAV, their fairness is the least. Therefore, the authors develop a method which controls and adjusts the UAV's speed according to the intra-cluster density of sensors, and the distance from the UAV and each sensor. However, unlike our this chapter which deals with multiple drones each with energy and storage constraints, the authors do not consider the energy consumption and the storage availability of the single drone in their approach.

The authors in [106] present an optimization problem where a drone is in charge to collect data from a set of ground sensors. Here, the locations where the drone has to stop are not known in advance, and they are computed according to a clustering scheme. The goal is to determine the single drone's path such that the drone's energy is minimized. When computing the path, a trade-off between the flight duration and the communication reliability is required. The problem

is solved relying on ILP formulations. A similar approach is proposed in [107], in which the goal is only to minimize the flight time of the single drone. The problem is optimally solved by performing a brute-force algorithm plus two heuristic algorithms. The above papers differ from our approach in that they only utilize a single drone, and they do not take into account data relevance in their modeling. Consequently, their objective is to maximize the quantity of collected data, whereas our approach is a more generalized version in which are involved multiple drones, and incorporated data relevance as a factor in our optimization.

In [108], the data collection problem is investigated under a "security" point of view. In fact, a fleet of drones is responsible for selectively collect data from a set of ground sensors, with the main goal to guarantee the security of the stored data. Different metrics are optimized in [108], such as computational cost, energy consumption, and communication overhead. The optimization function used in their approach did not take into consideration any kind of relevance of the data, as well as the storage of the drones.

The authors of [109] investigate data collection in a wireless sensor network by using multiple drones. Two main issues are addressed: (1) how to ensure seamless synchronization among sensors and drones at each transit, and (2) how to compute energy-efficient schedules for the sensors and feasible trajectories for the drones. To solve the problem, a joint wake-up scheduling and drone path planning optimization problem is formulated; eventually, simulations are also proposed. Similarly, in [110] drones are used to gather data from a set of ground sensors that rely on a Long Range Wide Area Network (LoRaWAN) protocol. The proposed problem takes into account three objectives, such as: (1) minimize the total drone's flying time, (2) collect all data packets from all nodes, and (3) minimize the nodes' energy consumption. Differently from our approach, the authors focus on prolonging the lifetime of the sensor network by only considering the limited battery capacity of the drones, without accounting for the storage constraint or the relevance of the collected data.

Finally, a recent work in [111] proposes an optimization problem to find a sequence of locations that the drone should follow inside the monitored area, so that the overall time to collect data is minimized. Unlike us, the authors only use a single drone and collect data from all sensors, without considering the relevance of their data or selecting a subset of sensors based on their relevance.

## 4.3 Problem Definition

In this section, we introduce the system model, define our novel problem for data collection in an IoT sensor network by leveraging drones, prove its *NP*-hardness, and devise the optimal solution for it by formulating an ILP.

### 4.3.1 System Model

Let $F$ be the *field*, whose center $O = (0,0,0)$ is the *depot*, to be monitored by a set $V = \{v_1, \ldots, v_n\}$ of $n$ heterogeneous IoT *ground sensors*. Each sensor $v_i \in V$ is randomly deployed in $F$, and its position is $(x_i, y_i, z_i)$ with respect to $O$. The field $F$ can be seen as a *complete graph* $G = (V, E)$ where the set $V$ is the set of vertices/sensors, and the set $E$ represents the edges/connections among each pair of sensors. The sensors collect data like the temperature, pressure, or even pictures or videos to be saved on their local storage of size $W_i$, assumed to be different for each sensor. In fact, tiny sensors that record text data have a small storage, but camera sensors that have to store large videos, need a much larger storage. Since $W_i$ is limited, sensors have to periodically transfer the recorded data to external devices. Let $0 < w_i \leqslant W_i$ be the *size* of the data that each $v_i$ needs to transfer. As mentioned above, the data are modeled by a *relevance*, and the relevant data should be prioritized when ground sensors have to start data transfer. This is modeled by associating a *reward* $r_i > 0$ to each sensor $v_i$. Data that need immediate analysis are prioritized with a higher reward than standard data, and data that may be lost, e.g, caused by any data overwritten, due to limited storage capacity are also associated with high priority. So, the more is the reward, the more relevant is to off-load the data to an external device. Importantly, given an instance of the problem, the relevance does not change, and remains the same until the mission is completed.

The external devices that collect sensor data are a set of $l$ homogeneous *drones* denoted as $D = \{d_1, \ldots, d_l\}$. The *flight mission* of each drone starts and finishes at $O$. The drones fly at a fixed altitude $h$ above the ground and have a *communication range* with a radius $R$. So, a drone $d_k \in D$ can collect data from a sensor $v_i$ if $\|d_k - v_i\|_2 \leqslant R$, i.e., if their relative *Euclidean distance* is within the communication range. In this chapter, we neglect communication issues such as shadowing, fading, or multipath propagation. Moreover, we assume that the drones and the sensors are always in line of sight and hence no obstacles are present. We also assume that the drones cannot collide among them or with any sensor. In fact, by opportunely tuning the heights at which drones fly, we can assume that drones cannot collide. However, for the sake of clarity, we simply consider the same height for all the drones even though collisions are still not addressed.

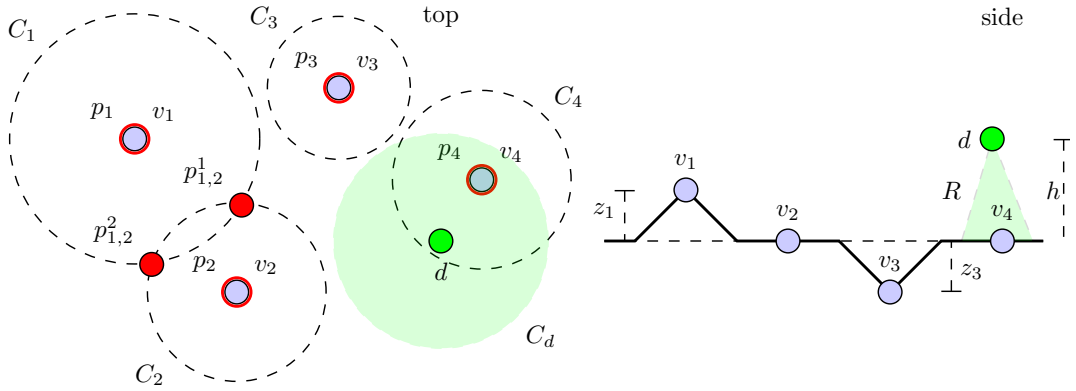



Figure 4.3.1: Top and side representations of the field $F$.

The drones are allowed to hover only at specific locations over $F$, called *waypoints*, represented by a set $P$ of possible positions. Firstly, $P$ contains all the positions of the projected sensors at height $h$, i.e., $\forall v_i = (x_i, y_i, z_i) \in V$ we have $p_i = (x_i, y_i, h) \in P$. $P$ also contains other points as follows. For each sensor $v_i$, we define an admissible region in which the drones can actually communicate with it. Such a region is delimited by a circumference $C_i$ of radius $\leqslant \sqrt{R^2 - (h - z_i)^2}$. To be more precise, a drone flying at a height $h$ has a communication sphere of radius $R$ which intersects the ground level ($h = 0$) forming a circumference $C_d$ of radius $\leqslant \sqrt{R^2 - h^2}$. A sensor can communicate with a drone if the center of $C_i$ is inside the circumference $C_d$. E.g., in Figure 4.3.1 the sensor $v_4$ and the drone $d$ can communicate. In general, the number of possible drone waypoints can be unbounded. Therefore, in order to bound the number of waypoints for the drone [100], for each pair of sensors $v_i$ and $v_j$, we add in $P$ all the possible intersections $p^1_{i,j}$ and $p^2_{i,j}$ between $C_i$ and $C_j$ (see Figure 4.3.1). Also the depot $O = p_0 \in P$. So, given the $n$ sensors, the number of waypoints is $m = |P| \leqslant n + n(n-1) + 1$, because two sensors can generate no more than two intersections.

Each individual drone is constrained by the limited energy of its battery that consumes when *moves* between locations and when *hovers* at waypoints. Sensors can start the data transfer procedure only when the drone hovers at waypoints. Hence, the sensors cannot transfer the data if a drone is currently moving even if their relative distance is within the communication range. We also assume that a drone cannot concurrently collect data from multiple sensors, but separately one at a time. So, if two sensors $v_1$ and $v_2$ are in range with a drone (say $d_a$), only one (say $v_1$) can transfer the data, while the other (say $v_2$) has to wait until $v_1$ finishes the procedure. However, if there is another drone (say $d_b$) in range with $v_1$ and $v_2$, and $v_1$ is transmitting towards $d_a$, $d_b$ could collect the data from $v_2$ at the same time because there is not any conflict nor any queue among them. Moreover, we do not allow for a partial transferring, so when a sensor $v_1$ starts to

transfer data to a drone $d_a$, $d_a$ necessarily must hover at the waypoint until the data is completely collected. This also means that a sensor $v_1$ cannot transfer a portion of data to a drone $d_a$, and the remaining portion to another drone $d_b$.

Given a waypoint $p_i \in P$, let $\mathcal{Q}_i$ be the set of ground sensors possibly in range with any drone, i.e., *covered* by a drone if it flies in $p_i$. So, each sensor $v_k \in \mathcal{Q}_i$ can communicate with a drone in $p_i$ because $\|p_i - v_k\|_2 \leqslant R$.

Let $\mathcal{E}^F(p_i, p_j)$ be the drone required *flying energy* for moving from $p_i$ to $p_j$, which depends on the Euclidean distance between waypoints, i.e., $\|p_i - p_j\|_2$, and on the *energy per unit distance* parameter $\alpha > 0$, so that:

$$\mathcal{E}^F(p_i, p_j) = \alpha \cdot \|p_i - p_j\|_2. \tag{4.1}$$

Let $\mathcal{E}^H(p_i, t)$ be the required drone's *hovering energy* for statically staying at the waypoint $p_i$ for $t$ time slots, which depends on the number of time slots, i.e., $t$, and on the *energy per time slot* parameter $\beta > 0$, so that $\mathcal{E}^H(p_i, t) = \beta \cdot t$. However, for a given $p_i$, the number of the needed time slots $t$ depends on the size of the data to be collected from the sensors in $\mathcal{Q}_i$. Recall that there are multiple drones, and a sensor can transfer its data to at most one drone. So, we have $\mathcal{Q}_i \supseteq \bigcup_{j=1}^{l} Q_i^j$, where $Q_i^j$ is the subset of sensors that the drone $d_j$ needs to collect the data from. Moreover, $\mathcal{Q}_i$ is a partition and hence we also have $Q_i^a \cap Q_i^b = \varnothing$ for any $a \neq b$. Thus, if there are sensors $v_k \in Q_i^j \subseteq \mathcal{Q}_i$, the cumulative data to transfer to the drone $d_j$ from the sensors in $Q_i^j$ at the waypoint $p_i$ is $\sum_{v_k \in Q_i^j} w_k$. So, the required time depends on the total quantity of data to be transferred, and on the *data-transfer rate* parameter $\gamma_k > 0$ for the sensor $v_k$, so that we can finally redefine the hovering energy function as:

$$\mathcal{E}^H(Q_i^j) = \sum_{v_k \in Q_i^j} \frac{\beta}{\gamma_k} w_k. \tag{4.2}$$

Let $\mathcal{M} = \{M_1, \ldots, M_l\}$ be the set of the $l$ drones' *missions*. Each individual mission $M_j$, accomplished by the drone $d_j$, is formed by a sequence of distinct waypoints to be visited to/from the depot $O = p_0$; i.e., $M$ is a sequence $p_0, \ldots, p_i, \ldots, p_0$. For each waypoint $p_i$, the drone $d_j$ actually obtains the data from a subset $Q_i^j \subseteq \mathcal{Q}_i$ of sensors due to its storage limitation. Finally,

$$\mathcal{C}_{M_j} = \mathcal{C}^F_{M_j} + \mathcal{C}^H_{M_j} \tag{4.3}$$

is the *total mission cost* (in terms of energy) of the mission $M_j$, where $\mathcal{C}^F_{M_j}$ is the *flying cost*, and $\mathcal{C}^H_{M_j}$ is the *hovering cost*. Notice that, the communication effort required for the drone is included as a function proportional to the time. Therefore, given the sequence of consecutive waypoints

$(p_i, p_k) \in M_j$, the flying and hovering energy costs are, respectively,

$$\mathcal{C}^F_{M_j} = \sum_{(p_i,p_k)\in M_j} \mathcal{E}^F(p_i, p_k) \tag{4.4}$$

$$\mathcal{C}^H_{M_j} = \sum_{p_i\in M_j} \mathcal{E}^H(Q^j_i) \tag{4.5}$$

The other aspect to consider is the overall transferred data from the sensors to the drones. Namely, let $\mathcal{U}_{M_j}$ be the *total used storage* by the drone $d_j$ when doing the mission $M_j$, i.e.,

$$\mathcal{U}_{M_j} = \sum_{p_i\in M_j, v_k\in Q^j_i} w_k. \tag{4.6}$$

Finally, let $\mathcal{R}_{M_j}$ be *total obtained reward* by the drone $d_j$ when doing the mission $M_j$, i.e.,

$$\mathcal{R}_{M_j} = \sum_{p_i\in M_j, v_k\in Q^j_i} r_k. \tag{4.7}$$

Note that, when $l = 1$, Eqs. (4.3) (4.6) (4.7) can be rewritten as $\mathcal{C}_M$, $\mathcal{U}_M$, and $\mathcal{R}_M$, respectively.

Concerning the drones' constraints, let $\mathbb{E} > 0$ be the *available energy budget* on the battery for performing a mission, i.e., each drone has its own energy budget $\mathbb{E}$ because the drones are assumed to be homogeneous. Moreover, let $\mathbb{S} > 0$ be the *available storage budget* on the mass storage for collecting the sensors' data. Again, each drone has its own storage budget $\mathbb{S}$. So, for each drone $d_j$, it is jointly required that:

$$\mathcal{C}_{M_j} = \mathcal{C}^F_{M_j} + \mathcal{C}^H_{M_j} \leqslant \mathbb{E} \tag{4.8}$$

$$\mathcal{U}_{M_j} \leqslant \mathbb{S}. \tag{4.9}$$

In this chapter, we assume that any mission formed by *only a single waypoint* is feasible for both energy and storage for a single drone. Specifically, all sensors in the vicinity of such a waypoint can safely offload their data to a drone.

### 4.3.2 The *Multiple-drone Data-collection Maximization Problem*

In this chapter, we present the *Multiple-drone Data-collection Maximization Problem* (MDMP) whose goal is *to find a set of routes for the drones to/from the depot, and a selection of sensors to assign to each drone, such that the sum of the total collected reward is maximized, and both the energy and storage budgets on each drone are not exceeded*. Given the set $V$ of $n$ sensors, the set $D$ of $l$ drones, and the energy and storage budgets of the drone $\mathbb{E}$ and $\mathbb{S}$, respectively, the objective is to determine the optimal set of missions $\mathcal{M}^*$ such that:

$$\mathcal{M}^* = \arg\max \sum_{d_j\in D} \mathcal{R}_{M_j} \ : \ \mathcal{C}_{M_j} \leqslant \mathbb{E},\ \mathcal{U}_{M_j} \leqslant \mathbb{S}. \tag{4.10}$$

Now, we are in a position to show that:

**Theorem 12.** *The* MDMP *is NP-hard.*

*Proof.* The classical Team Orienteering Problem (TOP), which has been proven to be *NP*-hard [112], can be seen as a particular instance of MDMP. Recall that in TOP the goal is to find a set of suitable closed routes for a given fleet of vehicles inside a weighted graph such that the sum of the total collected reward on visited vertices is maximized, and the traveling route cost of each vehicle along edges is within the given budget in input. However, in TOP, there is no storage constraint. So, we can reduce any instance of TOP to MDMP as follows: The total available storage constraint for a single drone can be relaxed by setting $\mathbb{S} = +\infty$. Concerning the reward, in TOP there is no choice to perform in each waypoint. This corresponds to assume that $Q_i^j = \mathcal{Q}_i$ in MDMP, i.e., we select for the drone $d_j$ all the reachable sensors at $p_i \in P$. After these modifications, any instance of TOP is exactly an instance of MDMP. So MDMP is *NP*-hard. When only one drone is considered, TOP becomes the Orienteering Problem (OP) which has been shown to be *NP*-Hard [112]. Hence, even for the single-drone scenario, MDMP is *NP*-Hard. □

In the next section, we will present the optimal algorithm capable of optimally solving MDMP.

### 4.3.3 ILP Formulation

The MDMP can be optimally solved using an ILP formulation. We enumerate the sensors as $\mathcal{V} = \{1, \ldots, n\}$, the waypoints as $\mathcal{P} = \{0, \ldots, m\}$ (0 is the depot), and the drones as $\mathcal{D} = \{1, \ldots, l\}$. Let $x_{ij}^k \in \{0, 1\}$ be a decision variable that is 1 if the sensor $i \in \mathcal{V}$ transfers its data to the drone $k \in \mathcal{D}$ at the waypoint $j \in \mathcal{P}$; otherwise, it is 0. Let $y_{\eta j}^k \in \{0, 1\}$ be a decision variable that is 1 if the drone $k \in \mathcal{D}$ travels from the waypoint $\eta \in \mathcal{P}$ to the waypoint $j \in \mathcal{P}$; otherwise it is 0. Let $1 \leqslant u_i^k \leqslant m$ be a dummy variable that indicates the temporal order of the waypoints visited by the drone $k \in \mathcal{D}$, i.e., $u_\eta^k < u_j^k$ waypoint $\eta$ is visited by the drone $k$ before the waypoint $j$ [8]. So, the ILP formulation is:

$$\max \sum_{k=1}^{l} \sum_{i=1}^{n} \sum_{j=0}^{m} r_i x_{ij}^k \tag{4.11}$$

subject to:

$$\sum_{k=1}^{l} \sum_{j=0}^{m} x_{ij}^k \leqslant 1, \qquad \forall i \in \mathcal{V} \tag{4.12}$$

$$\sum_{j=1}^{m} y_{0j}^k = \sum_{\eta=1}^{m} y_{\eta 0}^k = |\mathcal{D}|, \qquad \forall \eta, j \in \mathcal{P} \backslash \{0\}, \forall k \in \mathcal{D} \tag{4.13}$$

Table 4.3.1: Table of Notation.

| Symbol | Description |
|---|---|
| $F$ | field to be monitored by the drone |
| $V = \{v_1, \ldots, v_n\}$ | set of $n$ ground sensors |
| $(x_i, y_i, z_i)$ | position of sensor $v_i$ with respect to the center of the field $O$ |
| $G = (V, E)$ | complete graph representing $F$ |
| $E$ | set of edges/connections among sensors in $V$ |
| $W_i$ | local storage of sensor $v_i$ |
| $w_i \leqslant W_i$ | size of data that sensor $v_i$ needs to transfer |
| $r_i$ | reward associated with sensor $v_i$ (relevance) |
| $D = \{d_1, \ldots, d_l\}$ | set of $l$ drones |
| $h$ | altitude of the drones above the ground with respect to $O$ |
| $R$ | communication range radius |
| $P$ | set of possible drone waypoints |
| $p_i$ | a waypoint representing the position of projected sensor $v_i$ |
| $C_i$ | communication circumference of sensor $v_i$ |
| $p^1_{i,j}$, $p^2_{i,j}$ | intersection points between $C_i$ and $C_j$ |
| $m$ | number of waypoints in $P$ |
| $\mathcal{Q}_i$ | set of sensors possibly in range with a drone when flying in $p_i$ |
| $\mathcal{E}^F(p_i, p_j)$ | drone required flying energy for moving from $p_i$ to $p_j$ |
| $\alpha$ | energy per unit distance |
| $\mathcal{E}^H(p_i, t)$ | required drone's hovering energy for staying at $p_i$ for $t$ time slots |
| $t$ | number of time slots in which the drone hovers |
| $\beta$ | energy per time slot |
| $Q^j_i$ | subset of sensors that drone $d_j$ in $p_i$ needs to collect the data from |
| $\gamma_k$ | data-transfer rate for sensor $v_k$ |
| $\mathcal{E}^H(Q^j_i)$ | redefined hovering energy function |
| $\mathcal{M} = \{M_1, \ldots, M_l\}$ | set of the $l$ drones' missions |
| $\mathcal{C}_{M_j}$ | total mission cost (in terms of energy) of the mission $M_j$ |
| $\mathcal{C}^F_{M_j}$ | flying cost of mission $M_j$ |
| $\mathcal{C}^H_{M_j}$ | hovering cost of mission $M_j$ |
| $\mathcal{U}_{M_j}$ | total used storage by drone $d_j$ when doing the mission $M_j$ |
| $\mathcal{R}_{M_j}$ | total obtained reward by drone $d_j$ when doing the mission $M_j$ |
| $\mathbb{E} > 0$ | available energy budget on the battery for performing a mission |
| $\mathbb{S} > 0$ | available storage budget on the mass storage for collecting the sensors' data |
| $\mathcal{M}^*$ | optimal set of missions |

$$y^k_{jj} = 0, \qquad \forall j \in \mathcal{P}\backslash\{0\}, \forall k \in \mathcal{D} \tag{4.14}$$

$$\sum_{\eta=1}^{m} y^k_{\eta\nu} = \sum_{j=1}^{m} y^k_{\nu j} = \max_{i\in\mathcal{V}} x^k_{i\nu}, \qquad \forall \nu \in \mathcal{P}, \forall k \in \mathcal{D} \tag{4.15}$$

$$u^k_\eta - u^k_j + 1 \leqslant m(1 - y^k_{\eta j}), \qquad \forall \eta, j \in \mathcal{P}\backslash\{0\}, \forall k \in \mathcal{D} \tag{4.16}$$

$$1 \leqslant u^k_\eta \leqslant m, \qquad \forall \eta \in \mathcal{P}\backslash\{0\}, k \in \mathcal{D} \tag{4.17}$$

$$\sum_{i=1}^{n}\sum_{j=0}^{m} w_i x^k_{ij} \leqslant \mathbb{S}, \qquad \forall k \in \mathcal{D} \tag{4.18}$$

$$\sum_{j=0}^{m}\left(\sum_{i=1}^{n} h_i x^k_{ij} + \sum_{\eta=0}^{m} f_{\eta j} y^k_{\eta j}\right) \leqslant \mathbb{E}, \qquad \forall k \in \mathcal{D} \tag{4.19}$$

The objective function is represented by Eq. (4.11) which maximizes the overall reward. About the constraints, Eq. (4.12) states that each sensor can transfer its data no more than one time; Eq. (4.13) forces that the each drone's route begins and ends at the depot; Eq. (4.14) forbids self loops; Eq. (4.15) guarantees that each generated path is a simple cycle which contains the selected sensors; Eq. (4.16) ensures that no more than a single loop is allowed for each drone [8]; Eq. (4.17) indicates the temporal order of the visited waypoints, i.e., $u^k_\eta < u^k_j$ if $p_\eta$ is visited before $p_j$ by the $k^{th}$ drone [8]; Eq. (4.18) guarantees the storage constraint of each drone. In other words, the constraint guarantees that each drone cannot offload an amount of data larger than the storage capacity; Eq. (4.19) guarantees the energy constraint of each drone, where $h_i = \mathcal{E}^H(p_i, w_i) \geqslant 0$ is the drone's hovering cost for transferring the data from sensor $i$, and $f_{lj} = \mathcal{E}^F(p_l, p_j) \geqslant 0$ is the drone's flying cost for moving from waypoints $p_l$ to $p_j$.

We denote this formulation by OPT. Since OPT is only suitable for small inputs, in the following, we propose faster suboptimal algorithms suitable for any input. Specifically, in Section 4.4 we first devise algorithms for the particular case of a single drone, while in Section 4.5 we propose algorithms for the general case of multiple drones.

Table 4.3.1 summarizes the notation that has been adopted in this chapter.

## 4.4 Solving MDMP with a Single Drone

In this section, we propose an approximation algorithm, called *Reward-Storage-first Energy-then Optimization* (RSEO-S), and two greedy heuristic algorithms, called *Max ratio Reward-Energy* (MRE-S), and *Max ratio Reward-Storage* (MRS-S), respectively, in order to solve MDMP with a single drone, i.e., $l = 1$. Note that the suffix "-S" stands for "single-drone". Moreover, in this section we denote $M$ as the single-drone mission, while in the next section we will denote $\mathcal{M}$ as the set of missions of the fleet of drones.

### 4.4.1 The RSEO-S Algorithm

In this section, we devise an approximation algorithm that suboptimally solves MDMP with a single drone, called *Reward-Storage-first Energy-then Optimization* (RSEO-S). It is split into two phases. In the first phase, we select a subset of sensors such that the collected reward is maximized while ensuring that the storage requirement is met. Once the selection of sensors is done, we choose the minimum number of waypoints capable of covering all the selected sensors. At these waypoints, we compute the minimum energy-cost traveling route to/from the depot. Notice that the resulting route might be energy-unfeasible. If it is so, in the second phase we reduce the main route into a smaller one by removing the waypoint that reduces the least the lost reward. So, after the removal of the waypoint (along with two edges), we need to add an edge so that the path remains closed. This strategy is repeatedly done until we reach an energy-feasible route. The pseudocode of RSEO-S is given in Algorithm 13.

**Algorithm 13:** The RSEO-S Algorithm

```
1 V' ← knapsack(V, 𝕊)
2 P' ← min-set-cover(V', P)
3 M ← traveling-salesman(P')
4 while C_M > 𝔼 do
5     p ← arg min_{p_i ∈ M} R_M − R_{M\{p_i}}
6     M ← M\{p}
7 return M
```

The objective is to maximize the reward by considering the drone's storage, and initially neglecting the drone's energy. This is realized by approximating three classical *NP*-hard subproblems, namely the knapsack, the min-set cover, and the traveling salesman.

The RSEO-S algorithm works as follows. Let us now focus on the first phase. We initially determine a subset of sensors $V' \subseteq V$ such that the obtained reward is maximized and the storage constraint $\mathbb{S}$ is satisfied by invoking the *knapsack* procedure [113] (Line 1). In the knapsack procedure, we are given a collection of objects (sensors), each one associated with a size (data) and a reward (relevance), and we are asked to select a subset such that the total reward is maximized, while the total size occupied does not exceed that of the knapsack (drone's storage capacity). Let $V'$ be the set of selected sensors and $P$ be the family of subsets derived from the waypoints. Then, recalling that a sensor can be reached from multiple waypoints, we minimize the number of waypoints to visit to cover the entire set $V'$ by invoking the *min-set-cover* procedure [113], determining so a subset of $P' \subseteq P$ of waypoints with cardinality $|P'| \leqslant |V'|$ (Line 2). In the *min-*

*set-cover* procedure, we are given a set $V'$ (sensors) and a family of subsets on $P$ (waypoints), and the requirement is to select the minimum number of subsets whose union equals $V'$. Finally, since the drone has to perform a mission $M$ to/from the depot visiting the waypoints $P'$, we try to minimize the energy required by performing the *traveling-salesman* procedure [113] (Line 3). In the traveling-salesman procedure, we are given the set of points $|P'|$ (waypoints) in the Euclidean plane and a starting position $p_0$, with the requirement to traverse a tour starting and ending in $p_0$ so that all points are reached once and the traveled distance (energy) is minimized.

If the mission $M$ is energy-feasible, then $M$ is returned, otherwise we have to reduce $M$ by removing a vertex (along with two edges) during the second phase. From all the waypoints that form $M$, we remove the one that minimizes the loss of reward associated to the that waypoint (Line 5). When we remove such a waypoint, we need to add an edge that ensures the existence of a closed path to/from the depot. This is repeated until $M$ is energy-feasible. Eventually, the solution $M$ is returned (Line 7).

**Theorem 13.** RSEO-S *solves* MDMP *with a single drone with an approximation ratio of* $\frac{\psi}{\mu\phi}$ *where* $\mu\phi$ *is the number of waypoints returned by a* $\phi$*-approximation algorithm for the min-set-cover whose optimal solution has* $\mu$ *elements, which cover the sensors selected by a* $\psi$*-approximation algorithm for the knapsack.*

*Proof.* The solution $M$ depends on the computed cycle starting from the *knapsack* invocation (that returns a subset $V' \subseteq V$ of sensors), which in turn depends on the $\psi$-approximation algorithm for solving it [113]. Recall that, $\psi \leqslant 1$ because the *knapsack* is a maximization problem. Then, in order to reduce the number of waypoints from which we collect the data, we rely on a $\phi$-approximation version of the *min-set-cover* that returns at maximum $|P'| \leqslant \mu\phi \leqslant |V'| \leqslant n$ waypoints [113], where $\mu$ is the minimum number of waypoints able to cover $V'$. Recall that $\phi \geqslant 1$ because the *min-set-cover* is a minimization problem. If the resulting cycle given by the *traveling-salesman* [113] is energy-feasible, the approximation ratio of RSEO-S would be directly $\psi$, otherwise we need to prune some vertices reducing so the goodness of the solution. So, assuming $\mathscr{R}(SOL)$, $\mathscr{R}(OPT_{\text{KP}})$, and $\mathscr{R}(OPT)$ as the reward collected by the solution $M$, by the *knapsack*, and by the optimum algorithm, respectively, we can now prove that:

$$\mathscr{R}(SOL) \geqslant \frac{\psi}{\mu\phi}\mathscr{R}(OPT_{\text{KP}}) \geqslant \frac{\psi}{\phi\mu}\mathscr{R}(OPT).$$

The first inequality holds since we rely on a $\psi$-approximated solution provided by *knapsack*. Moreover, according to our assumption, the drone actually selects at least one waypoint. Therefore, by selecting the best waypoint among the $\mu\phi$ ones, the collected reward of $\mathscr{R}(SOL)$ is at least a

fraction $\frac{1}{\mu\phi}$ of the $\psi$-approximated solution of the *knapsack*. Finally, since $\mathscr{R}(OPT) \leqslant \mathscr{R}(OPT_{\text{KP}})$ is clearly true, the last inequality is also satisfied. □

In the next, we discuss the time complexity. In this chapter, we rely on the greedy strategy for *fractional knapsack* which requires $\mathcal{O}(n \log n)$ time and also guarantees a $\frac{1}{2}$-approximation [113], i.e., $\psi = \frac{1}{2}$. Recall that $|P'| \leqslant |V'| \leqslant n$. To implement *min-set-cover* we rely on a greedy strategy which takes $\mathcal{O}(m|V'|)$ (because $m$ is the cardinality of the subsets of sensors given by the waypoints) and guarantees a $(\log |V'|)$-approximation [113], i.e., $\phi = \log |V'|$. Regarding *traveling-salesman*, we exploit Christofides' $\frac{3}{2}$-approximation algorithm [113] (although it does not affect our ratio), which takes $\mathcal{O}(|P'|^3)$. Finally, since $M$ comprises of $\mathcal{O}(|P'|)$ edges, and considering that at each iteration we remove one vertex, the time required by the loop (Line 4) is $\mathcal{O}(|P'| \log |P'|)$. Thus, the overall time complexity of RSEO-S is $\mathcal{O}(n \log n + m|V'| + |P'|^3 + |P'| \log |P'|) = \mathcal{O}(n^3)$, and our approximation bound is bounded from below by $\Omega\left(\frac{1}{2\mu \log |V'|}\right)$.

### 4.4.2 The MRE-S Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with a single drone, called *Max ratio Reward-Energy* (MRE-S). MRE-S greedily adds, to the current solution the sensor whose ratio between the reward and the additional energy cost with respect to the current drone mission is the largest. When computing this ratio for any new sensor, we also have to consider the energy for going back to the depot, since the drone cannot remain without energy. Moreover, a sensor can be selected only if the current drone's residual storage is enough. The pseudocode of MRE-S is given in Algorithm 14.

**Algorithm 14:** The MRE-S Algorithm

```
1 M ← ∅, P̂ ← {p_0, p_1, ..., p_n} ⊆ P
2 while P̂ ≠ ∅ do
3     p ← arg max_{p_i ∈ P̂} r_i / (C_{M∪{p_i}} − C_M)
4     if C_{M∪{p}} ⩽ 𝔼 and U_{M∪{p}} ⩽ 𝕊 then
5         M ← M ∪ {p}
6     P̂ ← P̂ \ {p}
7 return M
```

Initially, the solution $M$ is empty, and a subset of waypoints $\hat{P}$ perpendicular to the sensors is created (Algorithm 14, Line 1). Then, the main cycle starts (Line 2) evaluating all possible waypoints $\hat{P}$. Among them, we select the waypoint $p_i$ whose sensor $v_i$ has the largest ratio between

the reward and the additional energy cost with respect to the current drone's mission (Line 3). This greedy selection is justified by the fact that we aim at maximizing the reward while trying to keep low the energy consumption. Then we evaluate whether $p$ can be added to the current solution $M$ without violating both the energy and storage constraints (Line 4). In any case, $p$ will not be considered anymore and removed from $\hat{P}$ (Line 6). Finally, the solution $M$ is returned (Line 7).

Regarding the time complexity of MRE-S, since the number of waypoints is $n+1$ (because we only considered waypoints perpendicular to the sensors), the main loop is repeated $\mathcal{O}(n)$ times (Line 2). Since the selection of the best waypoint (Line 3) takes into account at most $\mathcal{O}(n)$ waypoints in each iteration, the total time complexity of the MRE-S algorithm is $\mathcal{O}(n^2)$.

### 4.4.3 The MRS-S Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with a single drone, called *Max ratio Reward-Storage* (MRS-S). MRS-S, similar to MRE-S, is based on the highest reward-to-storage ratio (instead of reward-to-energy). The Line 3 in Algorithm 14 is replaced by $p \leftarrow \arg\max_{p_i \in \hat{P}} \frac{r_i}{w_i}$. Once the selection is done, MRS-S tries to add $p$ to the solution by evaluating if the energy and storage constraints are satisfied or not. In either cases, $p$ will not be considered anymore. Eventually, the solution is returned.

Unlike MRE-S, all ratios can be initially calculated once, and hence we can sort them in a decreasing manner, which takes $\mathcal{O}(n \log n)$. So, at each iteration of the algorithm, we extract the current best one in constant time. Therefore, its time complexity is $\mathcal{O}(n \log n + n) = \mathcal{O}(n \log n)$.

## 4.5 Solving MDMP with Multiple Drones

In this section, we extend the previous three algorithms for the single-drone case (in Section 4.4) to multiple-drones to solve MDMP in the general case, specifically the heuristic algorithms RSEO-M, MRE-M, and MRS-M, respectively. Moreover, we present two heuristic algorithms called *Span And Split* (SAS-M), and *Clusterize And Assign* (CAA-M). Note that the suffix "-M" stands for "multiple-drone".

### 4.5.1 The RSEO-M Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with multiple drones, called RSEO-M. It extends the RSEO-S algorithm to a multiple-drone scenario, so they

work similarly. In fact, the idea is to initially select and assign a subset of sensors to each drone such that the sum of the total collected reward is maximized while ensuring that the storage requirement is met (neglecting so the energy). Furthermore, as for RSEO-S, for each drone, we reduce the number of required waypoints to cover its assigned subset of sensors, then we compute a tour that connects all the waypoints and eventually shrink that tour if it is energy-unfeasible. The pseudocode of RSEO-M is given in Algorithm 15.

**Algorithm 15:** The RSEO-M Algorithm

1 $\mathcal{M} \leftarrow \varnothing$
2 $\mathcal{V}' = \{V'_1, \ldots, V'_l\} \leftarrow$ *multi-knapsack*$(V, \mathbb{S}, l)$
3 **foreach** $V'_j \in \mathcal{V}'$ **do**
4  $P'_j \leftarrow$ *min-set-cover*$(V'_j, P)$
5  $M_j \leftarrow$ *traveling-salesman*$(P'_j)$
6  **while** $\mathcal{C}_{M_j} > \mathbb{E}$ **do**
7  $p \leftarrow \arg\min_{p_i \in M_j} \mathcal{R}_{M_j} - \mathcal{R}_{M_j \setminus \{p_i\}}$
8  $M_j \leftarrow M_j \setminus \{p\}$
9  $\mathcal{M} \leftarrow \mathcal{M} \cup \{M_j\}$
10 **return** $\mathcal{M}$

The RSEO-M algorithm works as follows. Initially, the solution is empty (Algorithm 15, Line 1). Then we determine a collection of subsets of sensors $\mathcal{V}' = \{V'_1, \ldots, V'_l\}$ such that the sum of the reward obtained from each $V'_j$ is maximized and the storage constraint $\mathbb{S}$ is satisfied, by invoking the *multi-knapsack* procedure [114] (Line 2). The multi-knapsack procedure is a generalization of the knapsack procedure extended to multiple knapsacks. After that, for each knapsack $V'_j$ (Line 3) assigned to the drone $d_j$, we reduce the number of required waypoints (Line 4), we connect all of them (Line 5), and possibly shrink the tour created if it exceeds the energy budget (Line 6). Eventually, the solution is returned (Line 10).

In the next, we discuss the time complexity. In the RSEO-M algorithm we need to invoke the *multi-knapsack* procedure at the beginning, which is an *NP*-hard problem [114]. In particular, Chekuri et al. proposed a $(1-\epsilon)$ polynomial-time approximation scheme (PTAS) which takes $n^{\mathcal{O}(1/\epsilon^8 \log(1/\epsilon))}$ time [115, 116]. However, in this chapter we decided to rely on the fast heuristic algorithm proposed by Martello et al. whose time complexity is $\mathcal{O}(ln^2)$ [117, 118]. The other sub-procedures, i.e., *min-set-cover* and *traveling-salesman*, are executed $l$ times, but on inputs smaller than those in the RSEO-S algorithm, and in the worst case, the loop in (Line 3) takes $\mathcal{O}(ln^3)$ time. In conclusion, the overall time complexity of the RSEO-M algorithm is $\mathcal{O}(ln^3)$.

### 4.5.2 The MRE-M Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with multiple drones, called MRE-M. It extends the MRE-S algorithm to a multiple-drone scenario, and therefore they work similarly. In fact, MRE-M greedily adds to the current drone's solution among the available $l$ ones, the waypoint whose ratio between the overall obtainable reward and the additional energy cost with respect to the current drone's mission is the largest. Basically, MRE-M sequentially invokes MRE-S for each drone at the residual waypoints. The pseudocode of MRE-M is given in Algorithm 16.

**Algorithm 16:** The MRE-M Algorithm

1 $\mathcal{M} \leftarrow \emptyset, \hat{P} \leftarrow \{p_1, \ldots, p_n\}$
2 **foreach** $d_j \in D$ **do**
3 $\quad M_j \leftarrow$ MRE-S$(\hat{P} \cup \{p_0\})$
4 $\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{M_j\}, \hat{P} \leftarrow \hat{P} \backslash M_j$
5 **return** $\mathcal{M}$

The MRE-M algorithm works as follows. Initially, the solution is empty (Algorithm 16, Line 1), and the set of current waypoints is also created. Then, for each drone $d_j$ (Line 2), we iteratively invoke the MRE-S algorithm on the current set of waypoints $\hat{P}$ (Line 3). Specifically, we sequentially return an energy- and storage-feasible drone's mission $M_j$ for each drone to/from the depot, and then we add it to the set of missions $\mathcal{M}$, as well as we update the remaining waypoints (Line 4). Eventually, the solution is returned (Line 5).

The MRE-S algorithm takes $\mathcal{O}(n^2)$ time. Therefore, since the MRE-M algorithm sequentially invokes MRE-M $l$ times, in the worst case, the cost of MRE-M is $\mathcal{O}(ln^2)$.

### 4.5.3 The MRS-M Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with multiple drones, called MRS-M. It extends the MRS-S algorithm to a multiple-drone scenario, and therefore they work similarly. It exactly works as the MRE-M algorithm, with one exception, i.e., the Line 3 in Algorithm 16 is replaced by $M_j \leftarrow$ MRS-S$(\hat{P})$.

Since MRS-S is repeated $l$ times, its time complexity is $\mathcal{O}(ln \log n)$.

### 4.5.4 The SAS-M Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with multiple drones, called *Span And Split* (SAS-M). The idea is to initially create a path that connects all the vertices (sensors) plus the depot. This path is nothing but the sequence of vertices obtained by visiting the minimum spanning tree, rooted at the depot itself. Such a path exists because we assume that any two vertices can be connected following the straight line that connects them in the Euclidean plane. Then, we split the obtained path into a set of energy- and storage-feasible tours to be assigned to the drones. The pseudocode of SAS-M is given in Algorithm 17.

**Algorithm 17:** The SAS-M Algorithm

```
1 M ← ∅
2 T ← min-spanning-tree(G)
3 N ← tree-visit(T, p_0)
4 while N ≠ ∅ do
5     L ← create-tour(N, p_0)
6     M ← M ∪ {L}, N ← N\L
7 M ← best-l-missions(M)
8 return M
```

The SAS-M algorithm works as follows. Initially, the solution is empty (Algorithm 17, Line 1). Then, we determine the minimum cost tree $T$ in terms of drone's traveling energy that connects all the sensors $V$ plus the depot $p_0$, by invoking the *min-spanning-tree* procedure (Line 2). In the *min-spanning-tree* procedure, since we are given a weighted graph $G$ with weights on edges that represent the energy cost of flying between any two sensors, the objective is to select the tree that spans all vertices while minimizing the total flying/traveling energy cost on the edges. Once the tree is built, we perform a *tree-visit* procedure, and a sequence of vertices $N$ is generated in output (Line 3). After that, we begin splitting the sequence of vertices $N$ in tours to/from the depot $p_0$ (Line 4). Specifically, through the *create-tour* procedure (Line 5), starting every time at the point $p_0$, we try to create a tour $L$ by selecting the next available vertex $v \in N$ not already taken in other tours. A vertex $v$ can be added to the current tour $L$ if the travel cost to visit $v$ and go back to $p_0$, plus the hovering cost at $v$, is within the energy budget (as well as the whole collected data is within the storage budget); otherwise, the current tour $L$ is closed (Line 6). When $N = \emptyset$, we assign the best $l$ tours in terms of the collected reward to the $l$ drones (Line 7), and eventually we return the solution (Line 8).

In the next, we discuss about the time complexity. The *min-spanning-tree* procedure that

we have used in this chapter is Kruskal's implementation and requires $\mathcal{O}(|E|\log n)$ time [66], where $|E|$ is the number of edges of the (complete) graph. The *tree-visit* procedure visits the minimum spanning tree $T$. In this chapter, we implemented the Depth-first search (DFS) and the Breadth-first search (BFS): both take time $\mathcal{O}(n)$ on trees [65]. Now, the cycle in Line 4 sequentially considers the vertices given by the *tree-visit* procedure, one at the time. The vertices are $n$. Whenever a vertex is not anymore neither energy- nor storage-feasible, the current tour is closed (*create-tour* procedure). Hence, the cycle in Line 4 costs $\mathcal{O}(n)$. Finally, the selection of the best $l$ submissions costs $\mathcal{O}(n\log n)$ due to the sorting procedure. In conclusion, the SAS-M algorithm costs $\mathcal{O}(|E|\log n + n + n\log n) = \mathcal{O}(n^2\log n)$ since $|E| = \mathcal{O}(n^2)$.

### 4.5.5 The CAA-M Algorithm

In this section, we devise a heuristic algorithm that suboptimally solves MDMP with multiple drones, called *Clusterize And Assign* (CAA-M). The idea is to initially split the set of vertices into $l$ partitions (one for each drone) trying to balance the total energy cost. Then, we make each partition energy- and storage-feasible by invoking the previous RSEO-S algorithm. The pseudocode of CAA-M is given in Algorithm 18.

**Algorithm 18:** The CAA-M Algorithm

```
1 M ← ∅
2 V' = {V'_1, ..., V'_l} ← clusterize(V, l)
3 foreach V'_j ∈ V' do
4     M_j ← RSEO-S(V'_j, 𝔼, 𝕊)
5     M ← M ∪ {M_j}
6 return M
```

The CAA-M algorithm works as follows. Initially, the solution is empty (Algorithm 18, Line 1). Then we split the set of sensors $V$ into $l$ partitions by performing the *clusterize* procedure (Line 2). The clusterization phase takes into account the energy cost, so the hope is to determine clusters that are immediately energy-feasible regardless of the storage constraint. In other words, each cluster should be considered a neighborhood. This is in contrast to the RSEO-M algorithm, in which the "clusterization" given by the *multi-knapsack* procedure guarantees to have storage-feasible tours, while neglecting their energy cost. Therefore, each cluster $V'_j$ should be adequately reduced in terms of storage, and shrunk in terms of energy cost, through the RSEO-S algorithm (Line 4). Finally, the solution is returned (Line 6).

In the next, we discuss about the time complexity. The *clusterize* procedure is an implementation of k-means clustering, which aims to minimize the sum of the squares of the distances between each point and its closest center by locating $k$ cluster centers. This can be achieved through various methods, including Lloyd's local search algorithm, MacQueen's algorithm, and Hartigan-Wong's algorithm [119]. In this chapter, for its implementation easiness, we utilized Lloyd's implementation through the *clusterize* procedure, which has a time complexity of $\mathcal{O}(nl)$ [120].

Then, recalling that the RSEO-S algorithm costs $\mathcal{O}(n^3)$, the cost of running it $l$ times is $\mathcal{O}(ln^3)$. So, the final time complexity of the CAA-M algorithm is $\mathcal{O}(ln^3)$.

## 4.6 Performance Evaluation

In this section, we evaluate the performance, in terms of the reward obtained, of the algorithms presented to solve MDMP. We implemented[1] our algorithms in Python language version 3.9, and run all the instances on an Intel i7-10genK computer with 16GB of RAM. However, the ILP-based optimal OPT algorithm is implemented using IBM's ILOG CPLEX Optimizer solver v22.1 with Python used to wrap the objective and constraints and invoke the parallel solver.

In Section 4.6.2 we present the results with a single drone scenario, while in Section 4.6.3 we present the results with a multiple drone scenario. For the previous scenario, we compare the RSEO-S, MRE-S, and MRS-S algorithms with respect to OPT (the optimal one), while in the latter one, we compare the RSEO-M, MRE-M, MRS-M, SAS-M, and CAA-M algorithms with respect to OPT.

In Table 4.6.1, we compare the algorithms presented evaluating their time complexities.

### 4.6.1 Settings

The field $F$ is a square of the side 5km, where the depot is located at the center of it. We uniformly generate $n = \{10, \dots, 200\}$ sensors whose height is $-5\text{m} \leqslant z_i \leqslant 5\text{m}$. We also have a fleet of $l = \{1, 2, 3, 4\}$ drones that will be used to collect data. Each sensor has $100\text{MB} \leqslant w_i \leqslant 1\text{GB}$ data to transfer. This is a reasonable assumption because text data and picture data have different sizes in general. Moreover, each sensor has an associated reward $1 \leqslant r_i \leqslant 10$ that models the relevance of the data, so $r_i = 1$ models the lowest priority, while $r_i = 10$ models the highest priority. Both $w_i$ and $r_i$ are generated according to the Uniform distribution.

The drones fly at a fixed altitude $h = \{10, \dots, 45\}$m and have a fixed communication range of $R = 50\text{m}$ [121]. Their storage is $\mathbb{S} = \{2, \dots, 16\}$GB and their battery capacity is $\mathbb{E} = \{2.5, 5, 10\}$MJ [29].

[1]The code is available on GitHub here: https://github.com/TheAnswer96/JCSS

Table 4.6.1: Comparison between the algorithms for solving MDMP.

| | Algorithm | Section | Time Complexity |
|---|---|---|---|
| Single-drone | RSEO-S | 4.4.1 | $\mathcal{O}(n^3)$ |
| | MRE-S | 4.4.2 | $\mathcal{O}(n^2)$ |
| | MRS-S | 4.4.3 | $\mathcal{O}(n \log n)$ |
| Multiple-drone | RSEO-M | 4.5.1 | $\mathcal{O}(n^3)$ |
| | MRE-M | 4.5.2 | $\mathcal{O}(ln^2)$ |
| | MRS-M | 4.5.3 | $\mathcal{O}(ln \log n)$ |
| | SAS-M | 4.5.4 | $\mathcal{O}(n^2 \log n)$ |
| | CAA-M | 4.5.5 | $\mathcal{O}(ln^3)$ |

We fix an average energy consumption for flying $\alpha = 200\text{J/m}$ and hovering $\beta = 700\text{J/s}$ [122]. The data transfer rate between drones and sensors is set to $\gamma_i = 9\text{MB/s}$ (Wi-Fi 4 standard) regardless of the actual distance.

In the next plots, each algorithm is tested with different configurations of parameters, and we plot the average of the results on 33 random instances along with their 95% confidence interval. The optimal OPT is only run with small instances in input. In particular, when we run the OPT algorithm, we report in the $y$-axis the ratio $\rho = \frac{\mathcal{R}(SOL)}{\mathcal{R}(OPT)}$, i.e., the ratio among the total reward collected by the compared algorithm, and the total reward collected by the optimal algorithm. Clearly, $0 \leqslant \rho \leqslant 1$ because MDMP is a maximization problem. However, for larger instances in input, the OPT algorithm starts to be unsuitable due to the large number of variables/constraints, and therefore we only compare the other proposed approximation and heuristic algorithms. Obviously, since we do not compute the optimal solution, we report in the $y$-axis only the collected reward $\mathcal{R}(SOL)$ for each algorithm.

### 4.6.2 Results with a Single Drone

In this section, we evaluate our algorithms when solving MDMP with a single drone scenario. In Section 4.6.2, we assess the impact of drone altitude, while in Section 4.6.2, we assess the impact of energy and storage constraints.

#### Impact of the Altitude

In this section, we evaluate the impact of the altitude of the single drone.

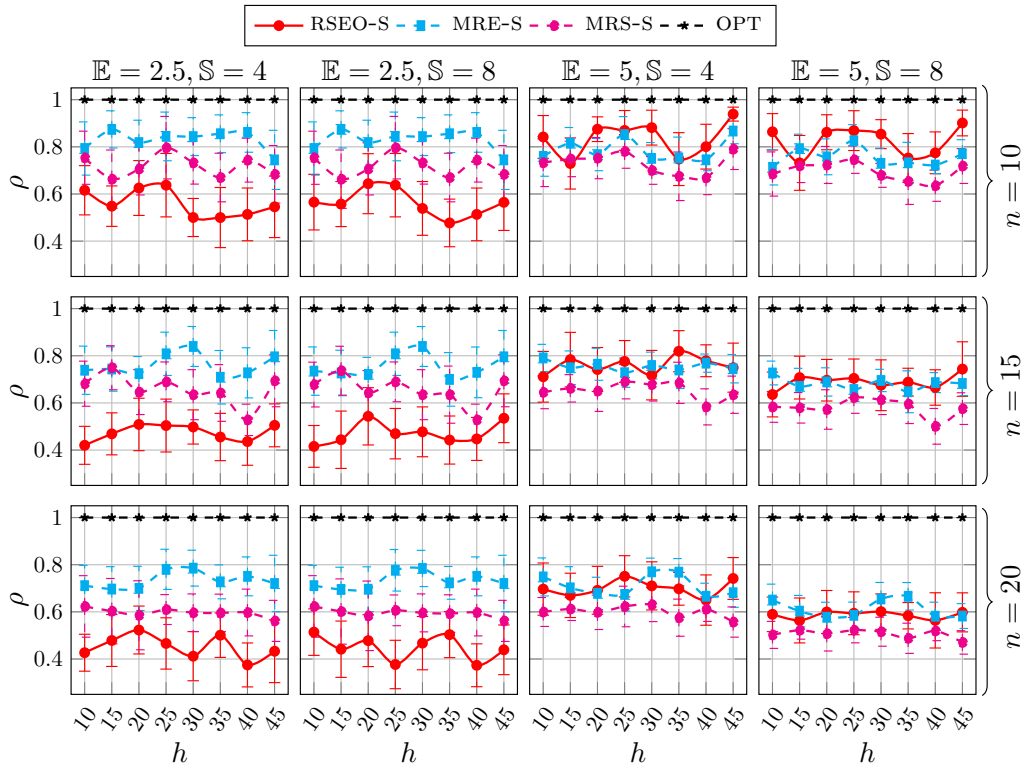



Figure 4.6.1: Single-drone: comparison of all the algorithms when varying the drone's altitude.

In Figure 4.6.1, we vary the altitude of the drone $h = \{10, \ldots, 45\}$m in the $x$-axis, while we report the ratio $\rho$ in the $y$-axis. We fixed small instances with $n = \{10, 15, 20\}$ sensors, since we also executed the OPT algorithm. In particular, since $n$ is small, we have chosen small values for energy and storage constraints, i.e., $\mathbb{E} = \{2.5, 5\}$MJ and $\mathbb{S} = \{4, 8\}$GB, respectively, obtaining so 4 different combinations of energy-storage. The plots in Figure 4.6.1 are organized in 3 rows, i.e., the first row shows the result with $n = 10$, the second and the third show $n = 15$ and $n = 20$, respectively; while the 4 columns show the combinations of energy-storage mentioned above.

The first observation in Figure 4.6.1 is that, by fixing a particular energy-storage setting, the results slightly change when we vary the height of the drone, with a few exceptions. The altitude parameter affects the number of waypoints that the drone can consider. In fact, the higher is the drone's altitude, the less is the number of intersections among the sensors, and hence the less will be the actual number of waypoints. Regardless of the drone's height, we can see that the RSEO-S algorithm poorly performs when the energy budget is small ($\mathbb{E} = 2.5$MJ). This is due to the fact that its initial strategy is aimed at finding a good partition of sensors such that the storage constraint is met. The selected sensors can belong to much different sub-areas of the field, and therefore the energy required for visiting all of them could be not sufficient. When the energy is doubled ($\mathbb{E} = 5$MJ), however, the drone has much higher chances of finding a suitable route that

can cover all the selected sensors and for this reason the RSEO-S algorithm performs better.

It is also interesting to see a counter-intuitive behavior with regard to the storage constraint. In fact, when the energy is small ($\mathbb{E} = 2.5$MJ), the more storage availability does not affect the results, neither positively nor negatively. This can be justified by the fact that when the drone has a limited battery, the resulted mission cannot be very long, and hence the larger availability of the storage is not detrimental. Instead, when the energy is larger ($\mathbb{E} = 5$MJ), a larger storage ($\mathbb{S} = 8$GB) slightly worsens the performance in terms of ratio. This is probably because the drone can visit more sensors flying longer routes, but the more availability of storage is not optimally exploited by the suboptimal algorithms.

The two greedy heuristic algorithms MRE-S and MRS-S perform more or less the same, with MRE-S that always outperforms MRS-S when varying the drone's height. When the energy is small ($\mathbb{E} = 2.5$MJ), the gap between the two strategies is more evident. In fact, MRE-S considers the energy budget when it builds the drone's mission. However, the time complexity of MRS-S is the lowest among all compared algorithms and often shows good enough performance.

In conclusion, since the height of the drone does not heavily affect the performance of our proposed algorithms, in the following, we consider a reasonable drone's height fixed to $h = 20$m.

#### Impact of the Energy and Storage Constraints

In this section, we evaluate the impact of energy and storage constraints for a given drone's altitude, i.e., $h = 20$m. In particular, we considered many more sensors in this comparison and therefore we do not execute the optimal OPT algorithm when $n \geqslant 25$.

In Figure 4.6.2, we fix the drone's altitude to $h = 20$m. We set the number of sensors $n = \{10, \ldots, 200\}$ on the $x$-axis, while we report the reward collected in the $y$-axis. Moreover, we set the drone's energy $\mathbb{E} = \{2.5, 5, 10\}$MJ and storage $\mathbb{S} = \{2, 4, 8, 16\}$GB. The plots in Figure 4.6.2 are organized in 3 rows and 4 columns, i.e., fixing the energy values on rows, and the storage values on columns.

In Figure 4.6.2 we can observe how the energy influences the performance of the algorithms. In particular, it is interesting to see the behavior when the energy $\mathbb{E}$ changes from 2.5MJ to 10MJ, specifically for $n = \{10, 15, 20\}$, i.e., the cases where we also performed the optimal OPT algorithm. Let us now focus on the case with $\mathbb{S} = 8$GB. In fact, when $\mathbb{E} = 2.5$MJ, the gap between the optimal algorithm and the other suboptimal ones is limited. Instead, when $\mathbb{E} = 5$MJ, such a gap is much more evident than before. However, when $\mathbb{E} = 10$MJ, the gap is reduced and, in addition, RSEO-S matches OPT. This is due to the fact that the RSEO-S algorithm can avoid pruning the sensors if the energy budget is very large.

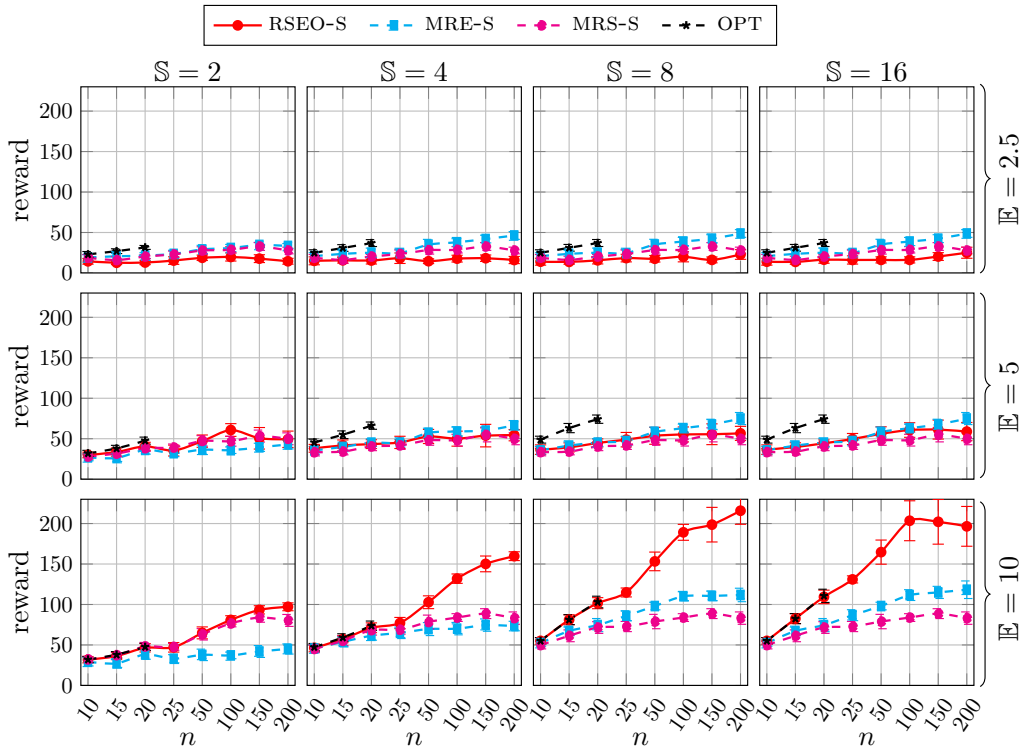



Figure 4.6.2: Single-drone: comparison of all the algorithms when varying the drone's energy and storage constraints.

Another interesting aspect to discuss is the collected reward when the number of sensors increases from $n = 100$ to $n = 200$. In fact, the total collected reward by the RSEO-S algorithm slightly decreases when $n = 200$. Recall that the strategy of RSEO-S is to consider the waypoints as a whole. Therefore, since the area of the field $F$ is the same, when the number of sensors increases, the density of them in $F$ also increases accordingly. As a consequence, the number of intersections dramatically increases and the hovering time on these will increase as well. Furthermore, during the pruning phase of RSEO-S, a certain number of waypoints, densely populated by sensors, will be discarded in order to keep the route within the energy constraint. Therefore, the final energy-feasible route will be sacrificed too much and there is a serious possibility that a residual energy budget can exist. In fact, we have experimentally observed that when $n = 200$, the residual non-used energy battery is a little bit larger than the cases when $n = 100$. This problem could be partially avoided by implementing a route reward-increasing phase in order to regain the unused energy, but this would not improve the guaranteed approximation ratio provided by the algorithm proved in Theorem 13.

In general, MRE-S outperforms MRS-S, especially when storage availability is high. Instead, for small storage in the input ($\mathbb{S} \leqslant 4$GB), the performance of MRS-S is better than that of MRE-S.

Despite its relatively poor general performance, the MRS-S algorithm is still worthy due to its lower time complexity with respect to that of MRE-S.

### 4.6.3 Results with Multiple Drones

In this section, we evaluate our algorithms when solving MDMP with a multiple-drone scenario.

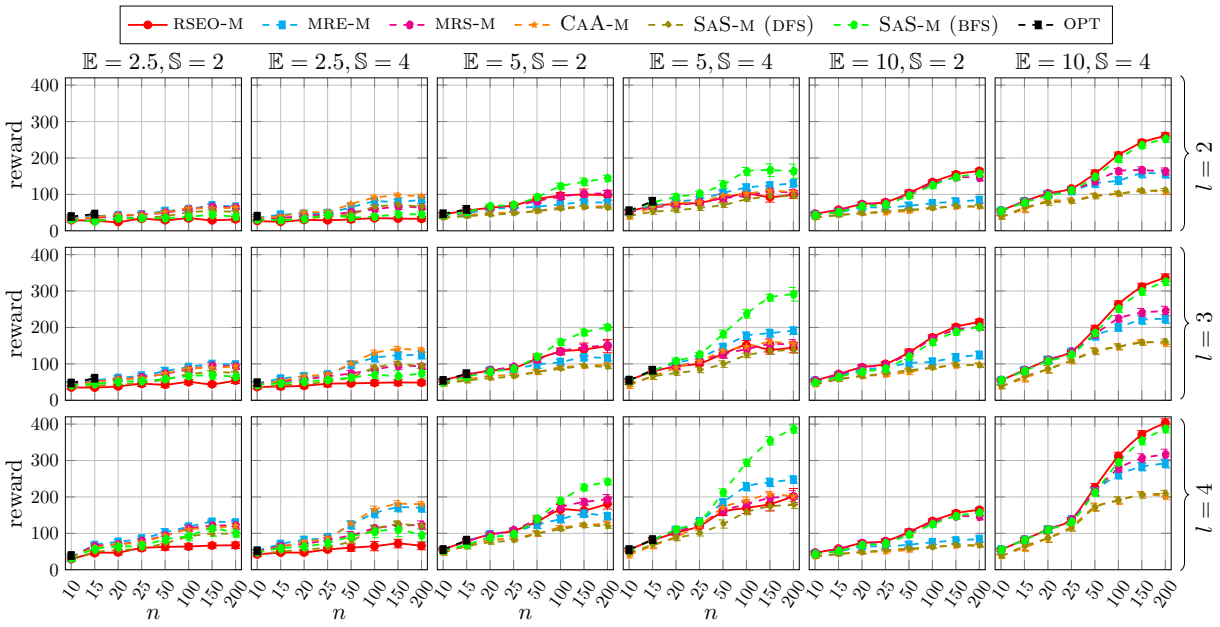



Figure 4.6.3: Multi-drone: comparison of all the algorithms when varying the drone's energy and storage constraints.

In Figure 4.6.3, we fix the drone's height to $h = 20$m. We set the number of sensors $n = \{10, \ldots, 200\}$ on the $x$-axis, while we report the collected reward on the $y$-axis. We set the drone's energy $\mathbb{E} = \{2.5, 5, 10\}$MJ and the storage $\mathbb{S} = \{2, 4\}$GB. The small values of the storage are justified by the fact that here we employ a fleet of $l = \{2, 3, 4\}$ drones, and hence drones can have less capabilities. The plots in Figure 4.6.2 are organized in 3 rows and 4 columns, i.e., fixing the number of drones on rows, and the energy-storage combinations on columns. Due to the large number of constraints, the optimal OPT algorithms has been executed only for the smallest instances.

The general trend that we can observe in Figure 4.6.3 is that when the energy is very small ($\mathbb{E} = 2.5$MJ), the best performing algorithm is CAA-M, while when the energy increases to $\mathbb{E} = 5$MJ, the best algorithm is SAS-M (when the BFS visit is implemented), and finally when the energy increases to $\mathbb{E} = 10$MJ, the best algorithm is RSEO-M. With respect to SAS-M, we can see a symmetric behavior with respect to the energy in the input. In fact, when the energy is

small, the DFS version of SAS-M outperforms the BFS one, while with more energy it happens the opposite. In the latter case, the gap between the two visit strategies is more evident. We can also observe that the performance of SAS-M increases when the number of drones $l$ also increases. Probably, this is justified by the fact that the DFS version tends to balance the distance of the sensors in each partition, whereas the BFS version is inclined to aggregate sensors into partitions with incremental distance from the depot. Therefore, when the budget is limited, the balanced nature of SAS-M DFS determines more profitable partitions, vice versa, it tends to "waste" energy by including very far sensors. The opposite reasoning can be applied for SAS-M BFS.

The RSEO-M algorithm's performance is poor when the energy level is low ($\mathbb{E} \leqslant 5$MJ). However, when there is more energy available ($\mathbb{E} = 10$MJ), the algorithm significantly improves. This trend is consistent with observations from the single drone scenario, where limited energy availability negatively affects the construction of an energy-feasible route for each drone. In contrast, if there is a large amount of energy available, the sub-sets of sensors returned through the *multi-knapsack* procedure become much easier to visit. Therefore, the algorithm's performance is dependent on the availability of energy, and it is more efficient when there is a higher energy level.

Finally, with respect to greedy heuristic algorithms, the MRE-M algorithm confirms to be a valid choice, although in the combination $\mathbb{E} = 10\text{MJ}, \mathbb{S} = 2\text{GB}$ its performance is not so good. This is because there is a high availability of energy, but a very low availability of storage. The other greedy solution, i.e., MRS-M, is almost always outperformed by MRE-M, but its time complexity is the lowest, and this is a good trade-off when the number of sensors and drones increases.

## 4.7 Conclusion

In this chapter, we investigated the problem of using a fleet of drones to collect data from IoT ground sensors deployed in a field to be monitored. As an example, in a smart agriculture scenario, a fleet of drones is in charge of retrieving data from many sensors to detect the presence of insects in orchards. The drones are constrained by both the available energy battery and the storage. The data that sensors have to offload to the drones is characterized by a size, and by a reward that models its relevance. The proposed problem is MDMP, whose goal is to plan a suitable set of missions for the drones such that the sum of the overall collected reward is maximized and the energy and storage constraints on each drone are both satisfied. We formally proved that MDMP is *NP*-hard, and presented an ILP formulation that optimally solves it. We also devised time-efficient approximation and heuristic algorithms capable of suboptimally solving MDMP.

In future work, we would like to incorporate communication issues between drones and sensors, as well as a more realistic environment with obstacles. Then, in order to evolve the MDMP towards a more realistic scenario and adjust the scheduling to user demands, we plan to consider a heterogeneous fleet of drones, taking into account the possibility to collect different amounts of data or offloading with diverse technologies and rates. Moreover, we can allow drones to fly at different altitudes during the same mission, for example, to cover more sensors or to improve air-to-ground communications. Finally, we plan to build a preliminary real test-bed with a single drone to collect data from real IoT sensors that store text and image information.

# Part III

# Drone and AI Algorithms for Pest Management/Detection in Orchards

# Chapter 5

# YOLO-based Detection of Halyomorpha halys in Images taken inside the Orchards

## 5.1 Introduction

In February 2021, the Horizon 2020 HALY.ID project [17] was granted with the objective of automating the monitoring activities by growers and plant health operators by July 2024. The main idea is to minimize or eliminate the reliance on traditional monitoring devices and activities, such as traps, visual sampling, sweep netting, and tree beating, while proposing an efficient and effective automated monitoring process for scouting the HH. Indeed, this chapter focuses on detecting the presence of HH. To achieve this, we start by creating a dataset of on-site images, primarily taken by drones as well as other devices. We enhance the dataset's quality by filtering out unsuitable images by evaluating blurriness and brightness parameters. Subsequently, we employ the YOLO framework to train several machine learning (ML) models using our curated dataset. These models are properly trained to recognize HH, and we assess their performance using various metrics, which yield positive outcomes. We conclude that by appropriately configuring the vision chip and optimizing the drone's positioning, issues of blurriness and brightness can be effectively mitigated and, under reasonable thresholds, are not limiting factors for the drone usage in bug detection.

### Contributions and Chapter Organization

Our results are summarized as follows:

- *We create the first dataset*[1] for HH using on-site images primarily captured by drones. We

[1] Currently, the dataset is kept private as per the HALY.ID consortium agreement. It can be only released on explicit

enhance the dataset's quality through a preliminary screening process aimed at removing noisy and biased images before conducting ML training. More specifically, we suggest using on-site images taken by the DJI Matrice 300 drone or other digital devices not only for testing but also for training purposes. Furthermore, to ensure confidence in image quality assessment, we evaluate blurriness and brightness using metrics from existing literature.

- *We train multiple ML models* using the YOLO framework[2] on our built dataset in order to detect the HH. When building our ML models, we apply three distinct approaches for training and validation that use three selections of images of different quality. We obtain satisfactory results by evaluating different metrics such as precision and recall, significantly improving on previous results.

- We develop a cloud-based service which can be reached from the field during drone mission for performing real-time detection of the HH.

The chapter is structured as follows: We present the relevant related work in Section 5.2. The description of the created dataset is given in Section 5.3, while Section 5.4 covers the conducted ML experiments. Finally, conclusions are drawn in Section 5.6. We further investigate in 5.A the influence of blurriness and brightness on image quality in the HH detection experiments.

## 5.2 Related Works

Drone have undergone significant advancements, particularly in agriculture, revolutionizing farming practices by delivering cost savings, operational efficiency, and increased profitability. Rejeb al. [124] conduct a thorough bibliometric review to consolidate and structure the academic literature on agricultural drones, unveiling current research trends. By employing bibliometric techniques, the analysis highlights key areas such as remote sensing, precision agriculture, deep learning, ML, and the Internet of Things as pivotal in the field of agricultural drones.

The utilization of drones in the field of agriculture can be advantageous in various ways. It can be used in conjunction with satellites to create vegetation indicators [125–127], or for monitoring wildlife and cows [128, 129], just to mention a few. The use of drones can result in the capture of a vast amount of imagery, and when combined with ML algorithms, it can make the system faster and more accurate than human observers in monitoring and estimating animal populations. Not

request to HALY.ID's Board. It will be made public for research purposes once the project concludes.

[2]We decide to employ YOLO as the object detector due to its excellent performance as demonstrated in [123] for HH detection in an artificial dataset.

only expensive drones, vision-based systems, and embedded electronics suitable for developed countries have been used for precision agriculture, but also low-cost solutions, such as AgriQ, proposed by De Oca et al. [130]. AgriQ comprises a drone, a multispectral imaging system, vision algorithms, autonomous trajectory planning, a budget-friendly multispectral imaging system, and open-source software to compute valuable information for farmers. Experiments show AgriQ's superior cost-effectiveness and performance compared to other commercial systems.

As for ML, there has been a growing emphasis on employing ML techniques for monitoring insect species. Traditional methods such as support vector machine, adaptive boosting, artificial neural network [131–134], and deep learning techniques based on convolutional neural networks (CNNs) [135–137] have demonstrated very good results in insect monitoring. For instance, a novel approach that involves the early detection and continuous monitoring of adult-stage whiteflies (*Bemisia tabaci*) and thrips (*Frankliniella occidentalis*) in greenhouses has been proposed in [132]. The approach is based on an image-processing algorithm and artificial neural networks. The developed identification algorithm achieves good results. This proposed approach has the potential to improve IPM strategies and reduce the use of harmful chemicals in greenhouse agriculture.

The chapter combines the use of new technologies, like drones with ML techniques to detect HH. To the best of our knowledge, only a limited number of studies have showcased the direct detection of insects through aerial surveys conducted with drones in open fields. In fact, drones can be equipped with specialized cameras capable of capturing high-resolution images of small objects, and can leverage GPS technology for efficient positioning. One of these studies aims to determine the effectiveness of drones in detecting the immobile stage of the *Monema flavescens* [138]. The results indicate that an aerial survey performed with a drone at a height of 3m above the tree canopy is more efficient and successful in identifying butterfly cocoons than a ground survey. Also, the captured images demonstrate the ability to differentiate between open and closed cocoons. So, the authors highlight the potential of drones for detecting insects directly in agriculture.

In the rest of this section, we present the state-of-the-art on HH detection. This section offers motivations for the achievements of this chapter, which finally improves the HH detection algorithms.

### 5.2.1 Previous Results on HH Detection

In the initial phase of the HALY.ID project, several attempts to scouting HH using different imaging technologies have been pursued. Ferrari et al. [139] evaluate the use of NIR-HSI as a potential

technology for detecting HH specimens on various vegetal backgrounds that can mimic field conditions. From a set of hyperspectral images comprising HH, two chemometric approaches have been used to develop classification models. The results demonstrate the potential of NIR-HSI combined with chemometric analysis and CNNs to detect HH accurately, even when mimicking different background conditions. Although this technology holds the potential to become an effective tool for IPM in the agricultural sector, it is currently not ready for field deployment due to its prohibitive cost. Additionally, several preliminary studies using RGB cameras have also been conducted to detect HH. Trufelea et al. [140] propose to train a CNN to recognize different insects, including HH, *Pyrrhocoris apterus*, and *Nezara viridula*. Most of the images are from the Maryland Biodiversity database [141], and others from a custom dataset built by professional cameras. Similarly, Ichim et al. [142] investigate the identification of HH with four CNNs. The dataset is built on two public datasets with many different insects specimen, plus a custom dataset collected with a DJI Mini 2 drone containing HH. The networks demonstrate strong performance only when evaluated on images resembling those in the public dataset. Finally, Sava et al. [123] assess the effectiveness of the YOLO [143] framework, employing region-based CNNs (R-CNN), for HH detection. Multiple models are trained, validated, and tested only using the Maryland dataset [141].

Table 5.2.1: Experimental results (*precision* among all the classes) on different testing datasets, as reported in [1].

| Row | Dataset | *P* |
|---|---|---|
| 1 | hh_unimore_rgb_lab-images_july2021_scaled | 0.97 |
| 2 | 2021-09-01-by-smartphone_scaled | 0.79 |
| 3 | farmer_scaled | 0.82 |
| 4 | drone-camera_2021-08-30_scaled | 0.28 |
| 5 | drone-camera_2021-08-31_scaled | 0.36 |
| 6 | drone-camera_2021-09-01_scaled | 0.30 |
| 7 | drone-camera_2021-09-03_scaled | 0.32 |

While the performance shown in [123, 140, 142] is indeed promising on Maryland dataset, the authors did not test their models on datasets constructed from on-site images or, if they did, they obtained poor results. So, the primary objective became the training of ML models on on-site images which is what any monitoring system should do. To do this, in our previous work in [1] we adopted a lightweight deep neural network (DNN) called *CenterNet* [144], trained solely on images captured by a DJI Matrice 300 drone in a first-person view mode. Unfortunately, the

performance was extremely unsatisfactory. To improve on the performance, an additional semi-artificial dataset was generated which included the silhouette and appearance of the HH extracted randomly from images with varying orientations and random backgrounds, sourced from public datasets (see Figure 5.2.1). After training CenterNet on the semi-artificial dataset, it was then tested on various other datasets whose results are reported in Table 5.2.1 [1]. The results indicate that the precision ($P$) is high for images captured with smartphones and even better for images obtained from professional cameras in the laboratory. However, the performance is considerably lower when using images taken with the drone. One could be tempted to brutally ascribe the poor results to the low resolution of the drone images. However, when observed with the naked eye, the images taken by the drone generally appear good, and even excellent in the portion of the photo where the bugs reside. Hence, we suspect that the decline in performance is primarily due to the disparity between the training and testing datasets.

(a) Silhouette on a background. (b) Drone's taken image.

Figure 5.2.1: The training dataset in [1].

**Motivations**. Considering the differences in results between [1] and [123, 140, 142], we attribute the variations in performance to the neural network used, and to the fact that we did not use the same image type for both training and testing like in [123, 140, 142]. Therefore, in this chapter we propose the utilization of on-site images taken by the DJI Matrice 300 drone or other digital devices not only for testing, but also for training. Moreover, to ensure confidence in the image quality and evaluate it, we assess the blurriness and brightness using metrics proposed in the literature. Specifically, we employ the no-reference blurring metric proposed in [145] and the brightness metric proposed by the International Commission on Illumination Laboratory (CIELAB) [146]. Our final goal is to substantially improve the performance of the HH detection algorithm, thus improving the rows of Table 5.2.1 that refer to drone images.

In the following section, we will present and explain the process we undertook to create the dataset utilized for the recognition of HH in the field.

At last, for easy reference, we report in Table 5.2.2 the list of acronyms adopted in this chapter.

Table 5.2.2: List of Acronyms and Descriptions.

| Acronym | Description |
|---|---|
| HH | *Halyomorpha halys* |
| NV | *Nezara viridula* |
| BMSB | Brown Marmorated Stink Bug |
| IPM | Integrated Pest Management |
| NIR-HSI | Near-Infrared Hyperspectral Imaging |
| ML | Machine Learning |
| DNN | Deep Neural Network |
| YOLO | You Only Look Once |
| CNN | Convolutional Neural Network |
| TL | Transfer Learning |
| IoU | Intersection over Union |

## 5.3 The New Dataset and its Analysis

In this section, we present the dataset that we created. We begin by explaining the composition of the dataset in Section 5.3.1. In Section 5.3.2 we investigate the blurriness and brightness of the selected images to objectively evaluate their quality. Finally, in Section 5.3.3 we examine the size of the bugs in the images, as this factor will play a crucial role in the labeling procedure.

### 5.3.1 Dataset Composition

One of the primary goals of the HALY.ID project was to develop a comprehensive and collaborative dataset of on-site images containing stink bugs. This dataset would serve as a valuable resource for computer vision algorithms to effectively identify the presence of the bugs in orchards.

During the first year of the summer campaign of the project, we acquired a total of 1,234 on-site images plus 34 high-quality images from the Internet. Table 5.3.1 reports the composition of the created dataset categorized in four different classes, namely, drone (see Figure 5.3.1a), smartphone (see Figure 5.3.1b), laboratory (see Figure 5.3.1c), and Internet images (see Figure 5.3.1d). As a result, our initial dataset consisted of a total of 1,268 images. From them, we discarded the images without HH. Furthermore, several images were excluded due to low quality

(a) On-site image with drone.

(b) On-site image with smartphone.


(c) Laboratory image.

(d) Maryland dataset image.

Figure 5.3.1: An example of image for each category.

issues. So, the final dataset has 677 images, out of which 83% where captured in the orchard, 12% in the laboratory, and 5% from other dataset in Internet.

The created dataset contains two classes of stink bugs: the invasive species, *Halyomorpha halys* (HH), and the most common stink bug in Italian orchards, *Nezara viridula* (NV), as shown in Figure 5.3.2. The HH and NV are quite different. From our dataset consisting of 677 images, there are 1,803 actual distinct instances of bugs, divided in 1,502 HH and 301 NV specimens. So, the scope of our algorithms is not only to detect a "stink bug", but also to classify it as either a HH or a NV. Although the classification is not a trivial task, the detection of the bugs is definitely an even more challenging task.

### 5.3.2 Evaluation of Blurriness and Brightness

As previously mentioned, the constructed dataset is heterogeneous as it has been assembled from diverse sources. Consequently, there is a significant amount of variability in the image quality, including variations in resolutions, aspect ratios, and other factors that could represent a bias for the computer vision algorithms, such as non-optimal blurriness and brightness [1]. Regarding the *blurriness*, some of the images taken during the drone's flight are out of focus due to the camera's

Table 5.3.1: Dataset composition.

| Class | Total | With HH | Used |
|---|---|---|---|
| Drone images | 855 | 653 | 289 |
| Smartphone images | 299 | 274 | 274 |
| Laboratory images | 80 | 80 | 80 |
| Total on-site | 1234 | 1007 | 643 |
| Internet images | 34 | 34 | 34 |
| **Total** | 1268 | 1041 | **677** |

Figure 5.3.2: Three Pentatomidae stink bugs: *Halyomorpha halys* (HH, left), *Nezara viridula* (NV, center), and *Rhaphigaster nebulosa* (RN, right, never observed in the monitored orchard).

auto-focusing mechanism. In fact, when shooting images at small distances, there is a possibility that the camera may fail to accurately focus at the intended distance, leading to out-of-focus images. Regarding the *brightness*, we have observed the presence of certain pictures with excessive exposure in certain areas or even throughout the entire image. This issue is likely attributable to the fact that the majority of the images have been captured during the peak hours of the day (late morning and early afternoon) in summer, when the sun is at its brightest. Additionally, it is not uncommon for the shadows created by the trees to cause an imbalanced distribution of light, resulting in excessive contrast along the edges of the image. So, with the purpose of avoiding biased training, a preliminary screening procedure has been applied by utilizing established no-reference estimators from the literature for both blurriness [145] and perceived brightness [146].

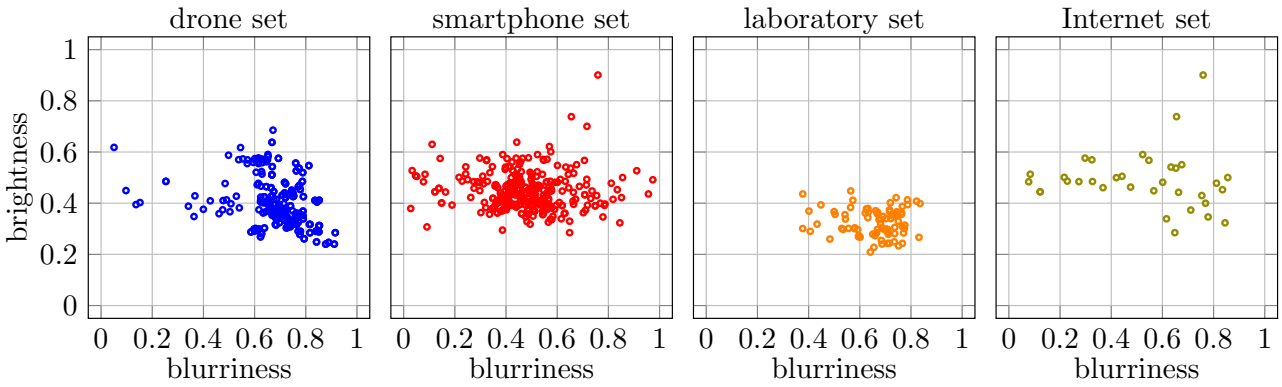



Figure 5.3.3: Dataset evaluation on blurriness, and perceived brightness for each acquisition source.

Figure 5.3.3 illustrates an analysis of the images in the dataset in terms of blurriness and perceived brightness according to the acquisition source. In detail, on the $x$-axis we list the blurriness scores, while the brightness scores are represented on the $y$-axis. The blurriness-metric is a score in the range $[0, 1]$: 0 represents blurred image, whereas 1 sharp image. Similarly, the brightness-metric is a score in the range $[0, 1]$: 0 represents under-exposition perception, whereas 1 means over-exposition. Hence, the images with the best metrics (blurriness closer to 1, brightness closer to 0.5) will stay in the middle-right portion of the plots.

The laboratory set (Figure 5.3.1c) is the most uniform according the two metrics. This is because the images have been taken in a light controlled environment, and keeping a fixed view of the target. The drone set (Figure 5.3.1a) reports higher values of brightness than the laboratory ones because the images have been taken outside. Nonetheless, we notice that overall the dots (images) are clustered together in a limited area with a few outliers due to the blurriness score. This is due to the fact that the images have been shot under similar distance and focal length

conditions. However, a few outliers exist because the drone's auto-focus can be inaccurate, as previously explained. The smartphone set (Figure 5.3.1b) exhibits the largest variance among blurriness scores. This is because the set comprises images taken by different operators, mainly farmers that helped in collecting images, using different devices, each equipped with diverse image sensors. Moreover, differently from the images collected with the DJI Matrice 300, no specific programmatic policy governed the image capture process. As for the Internet set (Figure 5.3.1d), it exclusively contains macro photos. In this case, the stink bug specimen occupies a significant portion of the image, and is the only element in focus. The low blurriness scores in this set occur when the stink bug occupies a limited area within the image. This is the only set that has quite high values for brightness, which seem to depend on the prevalence of white inside the picture. As a conclusion, as summarized in Table 5.3.2, the drone set offers on average good quality, also in comparison with the other sets.

Table 5.3.2: Dataset analysis and composition. The average (avg) is among all the pictures.

| Set | num. | blurriness | | | brightness | | | perc. |
|---|---|---|---|---|---|---|---|---|
| | | avg | min | max | avg | min | max | |
| Drone | 289 | 0.69 | 0.05 | 0.92 | 0.40 | 0.24 | 0.69 | 43% |
| Smartphone | 274 | 0.48 | 0.03 | 0.84 | 0.44 | 0.28 | 0.90 | 40% |
| Laboratory | 80 | 0.66 | 0.38 | 0.84 | 0.33 | 0.21 | 0.45 | 12% |
| Internet | 34 | 0.49 | 0.10 | 0.86 | 0.52 | 0.28 | 0.91 | 5% |
| Total | 677 | | | | | | | 100% |

### 5.3.3 Bounding Box Analysis and Labeling Phase

In Figure 5.3.4 (first row), we provide insights into the size distribution of the different bounding boxes for the two stink bug classes. The $x$-axis ($y$-axis) represents the width (height) in pixels of the drawn bounding boxes. In Figure 5.3.4 (second row), we show the distribution of the bounding box positions relative to the examined pictures. Essentially, these plots display the center position of each bounding box as a percentage with respect to the width and height of the picture.

Concerning the size distribution in Figure 5.3.4 (first row), since the dataset is heterogeneous, the dimension of the HH varies from, approximately, $30 \times 30$px to $1000 \times 1000$px. Differently, the NV samples have a limited bug size variance. Notice that the majority of images (97%) which contain NV have been taken using the drone, and the remaining 3% using smartphone cameras.

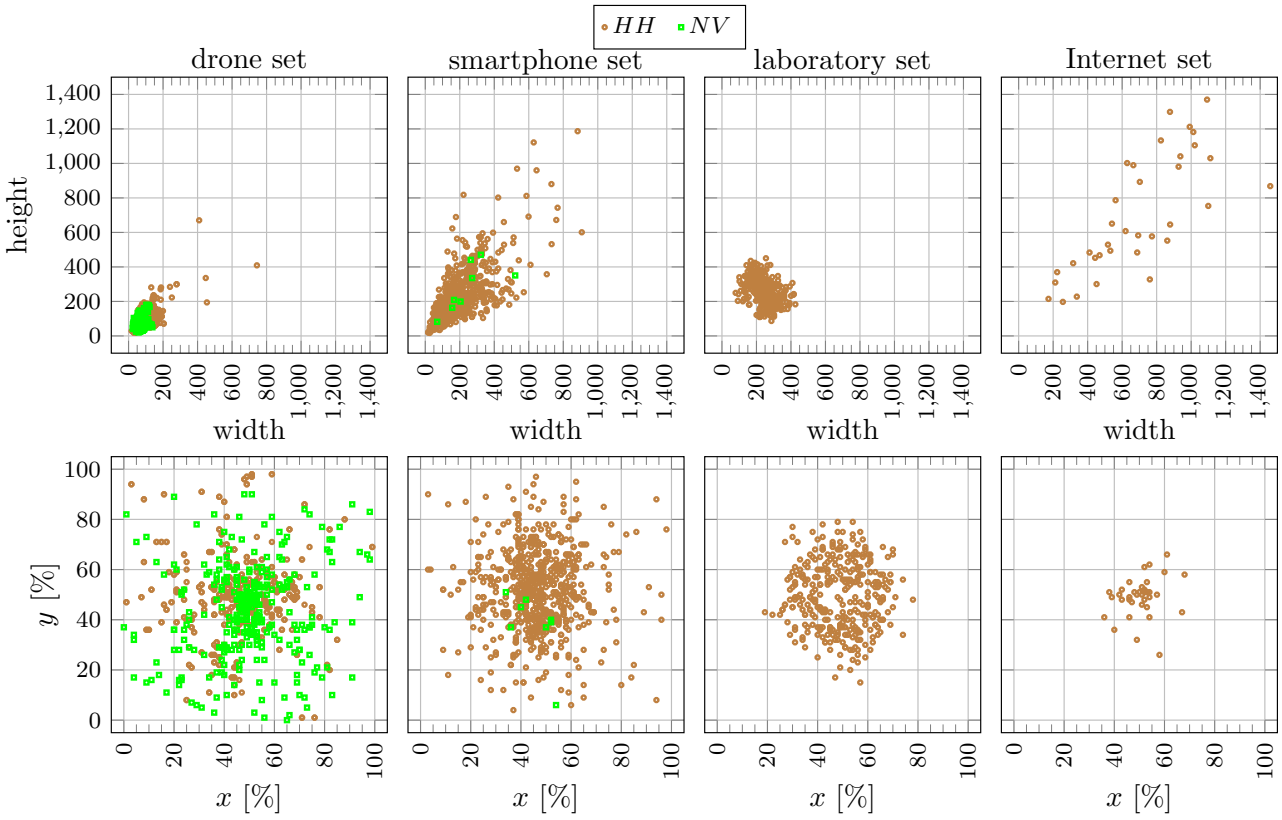



Figure 5.3.4: Bounding boxes analysis.

However, although there is a large variance in the bug size, more than 50% of instances are in the range of 200 × 200px. This is because the drone images have been taken with a strictly configured setting. Moreover, the drone set exhibits the smallest bounding boxes and the least variability in size. The smartphone set has a larger variability in the size, with bounding box sizes up to 1200 × 900px. We also find that the largest NV instances are in this particular set. However, we can note that the more is the bounding box size, the more rarefied are the instances in the plot. The laboratory set contains bounding boxes which are very similar with respect to their side sizes. Finally, considering the Internet set, we observe that it contains the largest bounding boxes of the built dataset regarding the HH. Indeed, this tiny set is characterized by high resolution macro images, where the bug represents the target of the shot.

Regarding the position distribution in Figure 5.3.4 (second row), the drone set highlights a strong variability for bug positions for both the classes. Even if the majority of the bugs approximately reside in the center of the images, there are some instances whose bounding boxes are located at the borders of the image. Since the position of the bugs in the orchard is unpredictable, the drone images capture a wider range of bug sizes and positions. A similar behavior is exhibited for the smartphone set. To conclude, both the laboratory and the Internet sets are characterized

by a very stable positioning of the bugs in which the bugs approximately cover the center of the image.

In the next section, we evaluate our computer vision algorithms from the created dataset in order to detect the stink bugs.

## 5.4 Performance Evaluation of the Bug Detection

In this section, we detail and discuss the performance evaluation of the HH detection. We give an overview of the adopted metrics for the comparison in Section 5.4.1. We then thoroughly compare and analyze the training and validation results in Section 5.4.2, and the testing results in Section 5.4.3.

For easy reference, Table 5.4.1 reports the used Symbols along this section.

Table 5.4.1: List of Symbols.

| Symbol | Description |
|---|---|
| $T_P$ | True Positive |
| $F_P$ | False Positive |
| $F_N$ | False Negative |
| $\tau$ | IoU threshold |
| $\gamma$ | Confidence parameter |
| $P$ | Precision |
| $R$ | Recall |
| $m_{0.5}$[3] | Mean Average Precision Pascal VOC challenge |
| $m_{0.95}$[4] | Mean Average Precision MS COCO challenge |
| BEST | Training set with the least blurred images |
| WORST | Training set with the most blurred images |
| RAND | Training set built without any policy |
| $\mathcal{S}$ | YOLO's small split |
| $\mathcal{M}$ | YOLO's medium split |
| $\mathcal{X}$ | YOLO's extra large split |

[3] Also known as $mAP[0.5]$.
[4] Also known as $mAP[0.5 : 0.05 : 0.95]$.

### 5.4.1 Adopted Metrics

YOLO detects the stink bug by returning a *prediction box* $B_p$ that "detects" the bug and its type. During the labeling phase, each bug is associated with a *ground truth box* $B_{gt}$. For each $B_p$, the IoU is defined as the ratio between the intersection area between $B_p$ and $B_{gt}$, and the union area between $B_p$ and $B_{gt}$, i.e., IoU $= \frac{\mathscr{A}(B_p \cap B_{gt})}{\mathscr{A}(B_p \cup B_{gt})}$, where $\mathscr{A}$ is the area. Note that if the prediction box detects a bug where there is none (i.e., the case of no-overlap with a $B_{gt}$), the area of $B_{gt}$ is assumed to be zero. A prediction box is considered a *detection box* if the IoU is above a threshold $\tau$. If not differently stated, we assume $\tau = 0.5$. We define:

1. **True Positive** ($T_P$): A correct detection (i.e., a prediction box with IoU $\geqslant \tau$) and object class identified.

2. **False Positive** ($F_P$): A wrong detection (i.e., a prediction box $B_p \neq 0$ with IoU $< \tau$). Note that this includes the case that the algorithm identifies a $B_p$ which does not overlap any $B_{gt}$. Namely, if $B_{gt} = 0$, we have IoU $= 0$.

3. **False Negative** ($F_N$): A ground truth not detected, i.e., missed detection $B_p = 0$. Since $B_{gt} > 0$, it holds that $B_p \cup B_{gt} > 0$.

Note that, since $B_{gt} \cup B_p \neq 0$, the denominator of IoU is always different from 0, and IoU is correctly defined.

Our experiments consider two stink bug classes: HH and NV. We measure the performance on the test set for each class, as well as for the combined performance of the two classes, by considering four metrics. Concerning the first two, we have precision ($P$) and recall ($R$), computed as follows:

$$P = \frac{T_P}{T_P + F_P}, \quad R = \frac{T_P}{T_P + F_N}, \tag{5.1}$$

where $T_P$, $F_P$, and $F_N$ are computed by fixing $\tau = 0.5$. Note that $T_P + F_N$ at the denominator of $R$ is the number of stink bugs, i.e., the number of ground truth boxes computed during the labeling process. Precisely, when $R$ refers to HH (respectively, NV), $T_P + F_N$ is the number of HH (respectively, NV) found in the training set. When $R$ refers to all classes, $T_P + F_N$ is the number of HH $\cup$ NV found in the training set.

Furthermore, we compute the mean *AP* $m_{0.5}$ (Pascal VOC challenge [147]) also known as $mAP[0.5]$, and $m_{0.95}$ (MS COCO challenge [148]) metrics, also known as $mAP[0.5 : 0.05 : 0.95]$). These metrics refine the notions of true-positive and false-positive by using the *confidence* parameter $\gamma$, i.e., a likelihood value returned by the network.

Fixed a confidence threshold $\tau_\gamma$, only the prediction boxes with $\gamma \geqslant \tau_\gamma$ are considered. A $T_P$ is then a prediction box with IoU $\geqslant \tau$ and $\gamma \geqslant \tau_\gamma$. Moreover, a $F_P$ is a prediction box with IoU $< \tau$ and $\gamma \geqslant \tau_\gamma$.

To compute the $m_\tau$ metric with $\tau = 0.5$, denoted as $m_{0.5}$, the refined precision and recall values are computed with the IoU threshold $\tau \geqslant 0.5$ and varying the confidence threshold $\tau_\gamma$. For each $\tau_\gamma$, only the prediction boxes that satisfy $\gamma \geqslant \tau_\gamma$ are considered. Then, among such images, the values $T_P$ and $F_P$ are determined based on the IoU condition. Consequently, the relative precision and recall values are recomputed. Then, a curve is built by plotting for each value of the recall (on the $x$-axis) the corresponding precision value on the $y$-axis and the approximated area of such a curve is returned as the mean average precision value $m_{0.5}$. The term "average" comes to the fact that the area of the curve refers to several values of confidence values. Note that when $\tau_\gamma$ decreases, the recall score cannot decrease because the $T_P$ cannot decrease and the denominator ($T_P + F_N$ ground truth) remains the same; while the behavior of precision is not predictable because both $T_P$ and $F_P$ cannot decrease, no claims can be made about the recall ratio.

Later on, in our discussion, we say that an object detector is *confidence-robust* if the precision is little affected by the variations of the confidence level. If so, it means that all the prediction boxes have a high confidence level, and the precision level remains almost constant. In other words, an object detector is robust if the $m_\tau$ score with $\tau = 0.5$, i.e., $m_{0.5}$ and the $P$ value, are close. The $m_{0.95}$ metric repeats the computation of $m_\tau$ by changing $\tau$ between $0.5 \leqslant \tau \leqslant 0.95$, with steps of 0.05; and returns the average of all the computed values $m_\tau$. From now on, we say that the detector is *IoU-robust* if the average precision is little affected by the IoU variations. If so, the actual IoU value of the prediction boxes is high (close to 1) for all the images. Hence, if the object detector is IoU-robust, the $m_{0.95}$ and $m_{0.5}$ scores are close.

In the rest of this section, we report first the metrics for the training and validation phases (Section 5.4.2) followed by those for the final HH detection on the testing set (Section 5.4.3). During this analysis, we consider three models trained with three different splits, BEST, RAND, and WORST. As we will see, the best results in training and validation will be obtained by the RAND model, thus confirming that the quality of the selected images is overall stable.

### 5.4.2 Training and Validation Results

Before testing the computer vision algorithms, a suitable training phase of the networks is required. The entire dataset has been split into three parts: a *training* set consisting of 407 images (60%), a *validation* set with 135 images (20%), and a *testing* set with 135 images (20%). In order

to investigate the claim that "the less blur an image has, the more accurate the prediction will be" [1], the 542 images from the both training and validation sets have been organized according to three different policies (BEST, WORST, and RAND) depending on a blurriness score [145]. Specifically, the least (respectively, most) 407 blurred images are selected and denoted as the BEST (respectively, WORST) training group, while in the RAND training group these images are simply randomly selected. Then, for each group, its validation set is built using the remaining 135 unselected images. These three groups will be used to train different object detectors, as explained below.


Figure 5.4.1: Example of image augmentation.

We also performed an *augmentation phase*. For each image, three new images obtained by random transformations are also included. Figure 5.4.1 shows an example of three different transformations such as mosaic enhancement and HSV-saturation (left), scale-in and transformation (middle), and combination of mosaic enhancement and scale-out (right). Since different images differ for their size, and since that TL uses weights obtained from the COCO dataset [149] which is composed by pictures with a size of 640 $\times$ 640px, we decided to scale down all the pictures to this particular value. This process is divided into two sub-phases: initially, the images are cropped in a square shaped eventually padded with white color, and subsequently scaled down to the fixed side length of 640px without losing the original ratios.

Each augmented training group will be used to validate and train an object detector based on a YOLO model. We trained the following versions of YOLO-v5, namely, small ($\mathcal{S}$), medium ($\mathcal{M}$), and extra large ($\mathcal{X}$). Furthermore, the models are trained by exploiting the pre-trained weights of COCO dataset, thus implementing the TL. We trained each model by setting a few parameters[5] such as batches with 32 pictures, 200 epochs, learning rate equals to $10^{-2}$, SGD optimizer [150], and image size of 640 $\times$ 640 pixels. Finally, nine object detectors are obtained because the $\mathcal{S}$, $\mathcal{M}$, and $\mathcal{X}$ YOLO models are trained with the three RAND, BEST, and WORST groups with TL

[5]For the training parameters, we refer to the YOLO's yaml file named *hyp.scratch-low* [143].

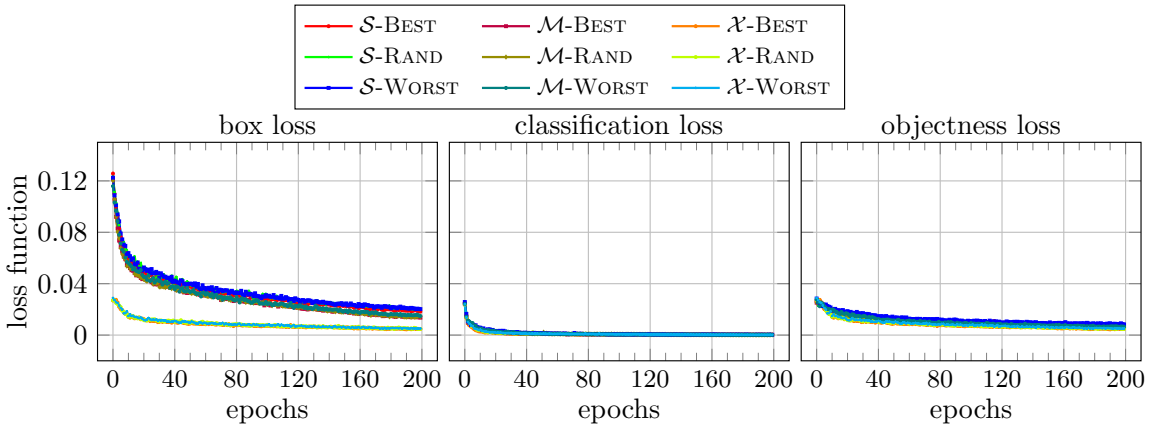
mechanism.

In Figure 5.4.2 we report the training and validation performance of the resulting YOLO models. For all the models and for all the training splits, Figure 5.4.2a shows the training performance in terms of loss functions, while Figure 5.4.2b shows the specific metrics used for the validation performance (i.e., $P$, $R$, and $m_{0.5}$). These are reported on the $y$-axis, while the epochs on the $x$-axis.

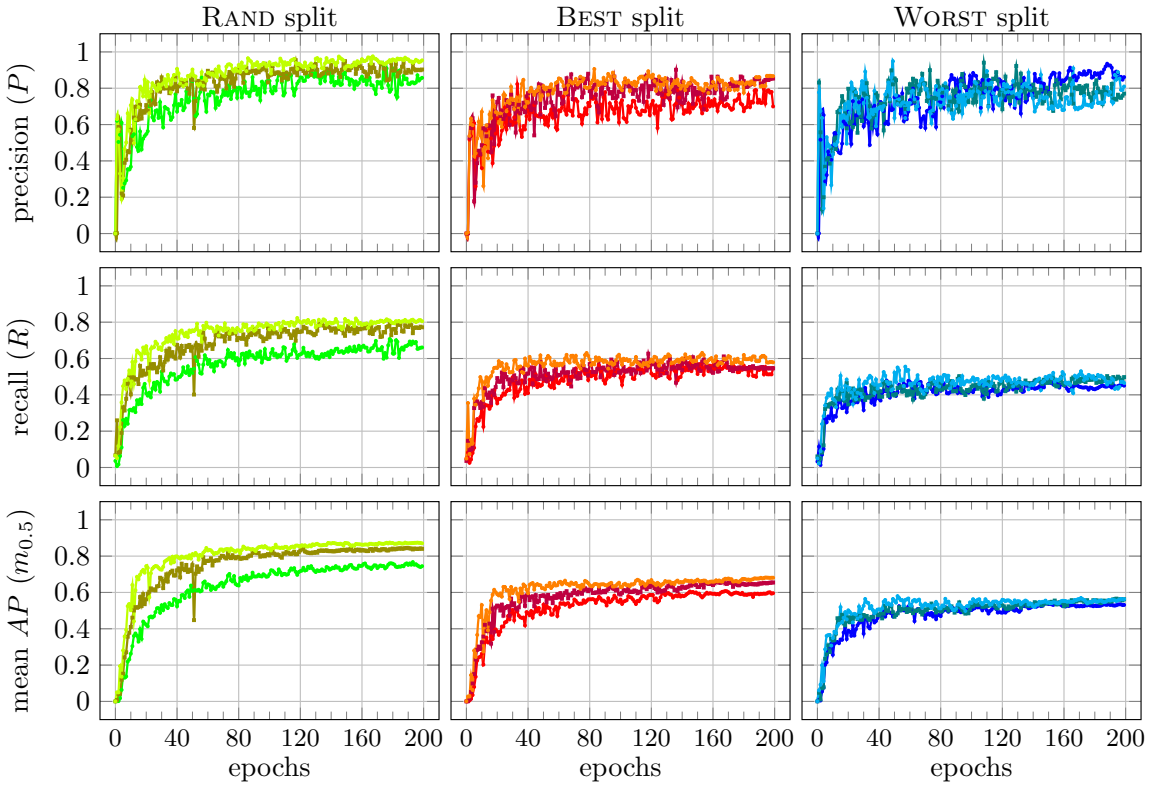
Concerning Figure 5.4.2a, in each of the three plots nine curves are shown: one curve for each combination of training split (RAND, BEST, WORST) and YOLO model ($\mathcal{S}$, $\mathcal{M}$, $\mathcal{X}$). The first plot reports the *box loss*, namely the function which establishes how well the model guesses bounding boxes coordinates. We observe that the models converge to a loss close to 0 fairly quickly. All the three training groups converge faster when the $\mathcal{X}$ model is used. The *classification loss*, which measures how well the model distinguishes between different classes, and the *objectness loss*, which is roughly speaking the confidence that some object exists in a given box, rapidly converge for all the models and all the groups. One can notice that the objectness loss, although it reaches a plateau, remains slightly higher than 0.

Focusing on Figure 5.4.2b, the three rows report the *precision*, *recall*, and *mean AP* achieved by the YOLO models during the validation phase (see Eq. (5.1)). The first column depicts the results of the three models trained on the RAND split, while the central and the last columns show the results for the BEST and the WORST split, respectively. Notice that the used metrics are values averaged on the two classes HH and NV to be recognized. In all the rows, the models trained on the RAND split reach better performance, faster than the models trained on the BEST and WORST splits, where the $\mathcal{X}$ is always the best. While the three groups behave almost the same for the $P$ metric, the $R$ and $m_{0.5}$ performance of WORST and BEST are worse than those of RAND. The $P$ curves of WORST and RAND fluctuate much more than those of $R$ and $m_{0.5}$.

Summarizing, we note that all the models obtain a satisfactory performance with both the loss functions and metrics. Also, they stabilize their trends reaching a plateau in 200 epochs; we can state that 200 epochs represent a reasonable trade-off between quality of the solution, and training time. Furthermore, we observe that the models trained on randomly selected images (RAND) reach the best results on all the metrics with a stable and smooth trend. The models trained on the WORST split exhibit the worst performance, while the BEST split allows the models to place in between the RAND and the WORST splits. Finally, focusing on the size of the models, we can notice that the larger is the model, the higher is the score, and this remark is valid for each metrics, and for each split considered.




(a) Loss function results.




(b) Precision, recall, and mean average precision metric results.

Figure 5.4.2: Training and validation results using blur score as splitting parameter.

### 5.4.3 Testing Results

Finally, we validate all the models on the testing set, without applying any transformations.

RAND split

| Mod. | HH Class | | | | NV Class | | | | All Classes | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ |
| $\mathcal{S}$ | 0.84 | 0.74 | 0.80 | 0.52 | 0.77 | 0.66 | 0.70 | 0.38 | 0.80 | 0.70 | 0.75 | 0.45 |
| $\mathcal{M}$ | 0.92 | 0.75 | 0.85 | 0.58 | 0.91 | 0.77 | 0.82 | 0.51 | 0.92 | 0.76 | 0.83 | 0.55 |
| $\mathcal{X}$ | 0.92 | 0.78 | 0.85 | 0.60 | 0.92 | 0.79 | 0.83 | 0.59 | 0.92 | 0.79 | 0.84 | 0.60 |

BEST split

| Mod. | HH Class | | | | NV Class | | | | All Classes | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ |
| $\mathcal{S}$ | 0.77 | 0.72 | 0.77 | 0.48 | 0.86 | 0.67 | 0.72 | 0.36 | 0.81 | 0.69 | 0.75 | 0.42 |
| $\mathcal{M}$ | 0.90 | 0.73 | 0.81 | 0.56 | 0.89 | 0.67 | 0.72 | 0.36 | 0.89 | 0.73 | 0.81 | 0.54 |
| $\mathcal{X}$ | 0.98 | 0.71 | 0.84 | 0.59 | 0.99 | 0.83 | 0.86 | 0.62 | 0.97 | 0.77 | 0.85 | 0.61 |

WORST split

| Mod. | HH Class | | | | NV Class | | | | All Classes | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ | $P$ | $R$ | $m_{0.5}$ | $m_{0.95}$ |
| $\mathcal{S}$ | 0.89 | 0.74 | 0.81 | 0.54 | 0.80 | 0.70 | 0.71 | 0.38 | 0.84 | 0.72 | 0.76 | 0.46 |
| $\mathcal{M}$ | 0.95 | 0.73 | 0.84 | 0.58 | 0.94 | 0.77 | 0.81 | 0.52 | 0.95 | 0.75 | 0.82 | 0.55 |
| $\mathcal{X}$ | 0.95 | 0.77 | 0.86 | 0.61 | 0.95 | 0.85 | 0.87 | 0.60 | 0.95 | 0.81 | 0.86 | 0.60 |

Table 5.4.2: Results of YOLO models trained on different images.

Table 5.4.2 reports the results on the testing sets. In general, looking at all the three groups, we can observe that the $\mathcal{M}$ and $\mathcal{X}$ models obtain a performance above 89% for all the classes. The $R$ metric scores for the same models varies between 73% and 85%. Both HH and NV classes are recognized with a good balance between $P$ and $R$. For the HH class and RAND split, $P$ is always above 92%, and hence approximately a tenth of the HH bugs is misclassified. However, $R$ is only above 75%, which indicates that the models tend to miss the HH achieving more $F_N$. The NV class has higher $P$ than HH, although obtains a worse performance on $R$. This means that the NV individuals are rarely misclassified, although they are often unrecognized.

Generally, the $m_{0.5}$ of the HH class is higher than that of the NV class, and the difference

between $m_{0.5}$ and $P$ is smaller for the HH class than for the NV class. This means that the models for the HH class are more confidence-robust than those for the NV class. Instead, the difference between the $m_{0.95}$ from $m_{0.5}$ is high for both the classes, meaning that there is a high variations in the IoU size of the prediction boxes. The results on the "ALL Classes" confirm the above results. The models are not IoU-robust because the $m_{0.95}$ and $m_{0.5}$ scores significantly differ. Instead, the models are relatively confidence-robust since the distance between $P$ and $m_{0.5}$ is moderate.

Looking at the metrics, there is a dominance of the $\mathcal{X}$ model, which confirms the behavior of the training. Namely, concerning the BEST split, it reaches the highest values for $P$ on both the classes HH and NV (98% and 99%, respectively) with the $\mathcal{X}$ model. However, with the same model, the BEST split reports the worst $R$ scores, except for the NV class. One possible interpretation is that when presented with very sharp images, the model can establish a strong understanding of the distinctive features of stink bugs, and it almost never misclassifies. However, it lacks of flexibility in the decision, and it misses several bugs.

The RAND split is the set of images that guarantees the most balanced $P$ and $R$ values. This is likely due to the random selection, which enables the networks to train on a more diverse set of samples. As a result, they acquire a higher level of generality that aids in the training process. As regard the WORST split, it surprisingly reaches really good results. In particular, we can observe that results of the $\mathcal{S}$ model dominate those of the $\mathcal{S}$ model with the BEST and RAND sets, but in general there is no dominance. We believe that these results simply confirm that there is not a marked gap between the best and the worst samples in our dataset, as shown in the previous Section 5.3.2). In other words, the quality of our dataset is on average good.


(a) Camouflage background. (b) Dense targets scenario.

Figure 5.4.3: The main models' pitfalls.

Finally, we complete our result analysis by pointing out the main limitations, in terms of performance, behind our proposed detection models. As previously said, the poorest metric is the $R$, i.e., how many bugs are effectively detected. We obtained low values such as 0.71 and 0.66 for

the HH and NV class, respectively. Figure 5.4.3, which depicts two images on which the models were tested, highlights the main pitfalls found during the testing process. Inside the images, in green are reported the correctly detected bugs, whereas in red are highlighted the missed ones, i.e., the $F_N$. One of the most relevant issue behind the $R$ deterioration is related to target specimens which camouflages with the background due to their color. In detail, Figure 5.4.3a clarifies this particular pitfall. Certainly, we can observe that the failure to detect HH is mainly because its color is excessively similar to that of the underneath branches. Moreover, its perspective does not allow to identify all the HH main features, e.g., the paws. As regard to the second most common pitfall, it is strictly related to the "region proposal mechanism" adopted by YOLO-v5. Indeed, the system divides the image into a grid. Each of these grid cells predicts a certain amount of bounding boxes and confidence scores. Then, a suppression procedure is applied to remove the duplicates based on IoU between boxes and confidence score. Due to this suppression strategy, YOLO struggles to detect small objects cluttered together in a limited portion of the image. As shown in Figure 5.4.3b, there is a group of HH particularly close to each other, and although the majority are quite visible at the naked eye, only a subset of them were correctly detected. The model probably misses a bunch of HH because of the YOLO's suppression algorithm. Concerning the $P$ metric, the very few encountered $F_P$ are strictly related again to the color issue. In particular, some spots on the tree branches very similar to the HH were recognized as such, but since the number of $F_P$ is very limited this issue is quite negligible.

In conclusion, we obtained a significant improvement in the results if we compare them with respect to the first approach reported in Table 5.2.1 (data from [1]). Specifically, our images are captured with the drone and thus our results are comparable with those reported in Rows 4-8 of Table 5.2.1. This achievement is due to the quality of our dataset [6] and the fact that both the testing and training sets comprise images captured under similar conditions.

## 5.5 Developed Client-Server Application

In this section, we present a client-server application that can be run in the orchard in order to detect the presence of bugs.

[6] In 5.A, we intentionally blur the images of the RAND split. We show that by increasing the percentage of blurred images on the RAND split, all the metrics deteriorate.

### 5.5.1 Overview Architecture

The main idea of the application is to build an *almost real-time* mechanism capable of detecting (as well as counting) the bugs in the field. We specify "almost" because we do not detect bugs from the live video stream, but only from immediately-after taken pictures by the clients. A client can be any device with an attached camera (such as smartphones, tablets, as well as drone) and network capabilities. Basically, the application comprises two components: a *server*, and a *client*. The server contains the trained models previously discussed in Section 1.5, and it listens to *requests* sent by one or more clients. On each request, the client encapsulates the captured images to be evaluated by one of the different ML algorithms stored inside the server. Once the server finalizes to process the request, it replies to the client by sending back the possible detected bugs (in the form of the found bounding boxes coordinates) along with the computed confidence.

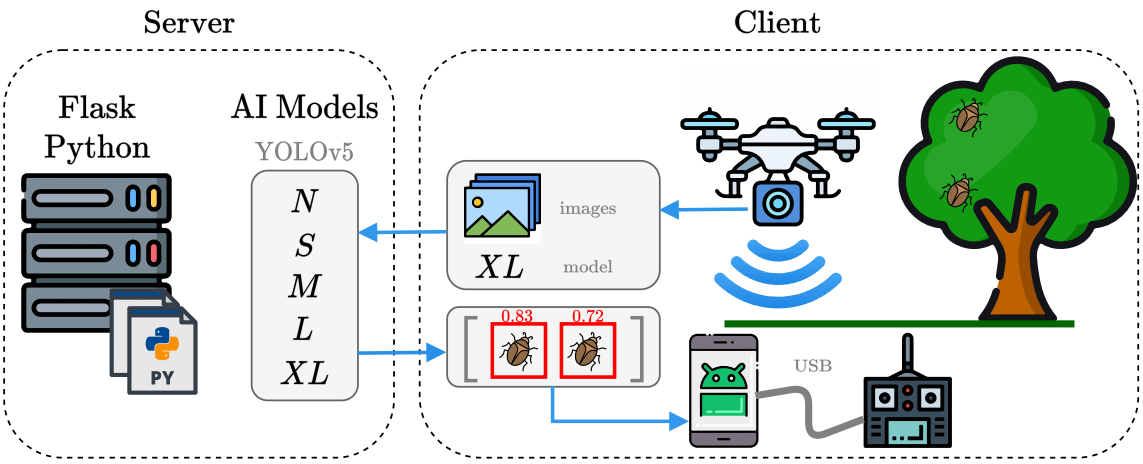



Figure 5.5.1: The app-architecture.

The architecture of the client-server application is shown in Figure 5.5.1.

### 5.5.2 Server Side

The server component is in charge of providing a stink bug detection to the clients that make requests. In other words, it remains passive until a client makes a request formed by a set of taken images, and a specific model to be queried. The server publicly exposes a Flask service, which is a micro web framework written in Python. Flask is a Python-based web framework that falls under the category of micro-frameworks as it does not demand any specific tools or libraries [151]. We relied on it simply for its simplicity, i.e., it does not require particular tools or libraries. The Flask service listens to incoming requests through HTTP POST requests. In fact, by using POST requests the client can enclose the captured images in the body of the request message, which is

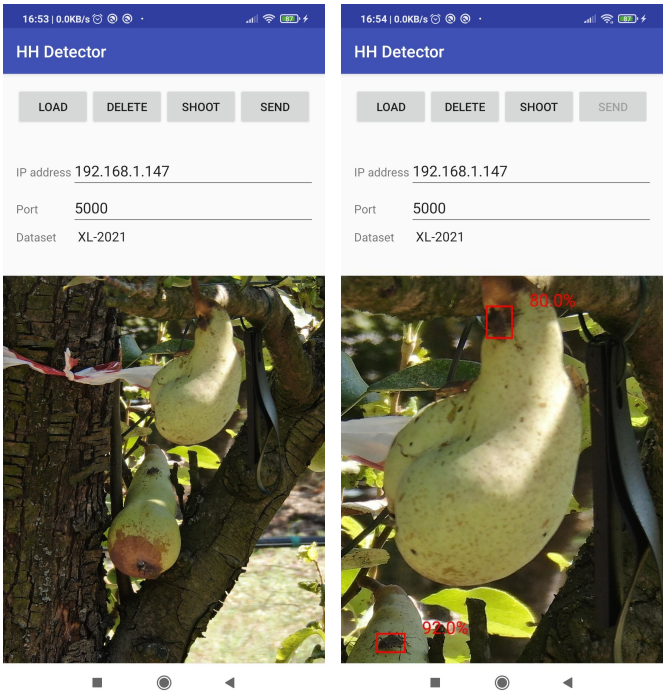
fundamental in this case. So, the client can perform an HTTP POST request through a particular URL (in the form of `url/model:port`) along with the enclosed images inside the message body. Concerning the `model` string inside the URL, we can use any of the models shown in Section 1.5 (e.g., `xl-blur-model` or `m-blur-model`). At this point, when the client triggers the particular exposed service, the server performs the image recognition procedure by relying on the trained models. This is done by PyTorch [152]. In the case that the model detects the presence of bugs, a set of coordinates that characterize the bounding boxes along with the computed confidence value are sent back to the client that made the request.

### 5.5.3 Client Side

The client component is in charge of taking pictures, and send requests to the server in order to detect the presence of stink bugs in the orchard. Due to the fact that the server component is decoupled in this architecture, different devices with different technologies can perform requests to the server. In other words, any device able to do HTTP POST requests. In the realm of the HALY.ID project, the client can be either a drone, smartphone, or tablet, or simply a laptop/computer. For smartphones and tablet we implemented an Android-based Java application that can be installed on Android devices, while for the latter ones we implemented an HTTP-based web application. In either case, the application can take/pick pictures and select a model to be queried via POST requests.

The application that runs on the DJI Matrice 300 has been written in Java language by leveraging the DJI Mobile Software Development Kit (MSDK) V4. The MSDK V4 allows to develop custom Android- or iOS-based applications to be installed on mobile devices directly connected to the drone's remote controller. With such applications the drone can perform automatic flights at different waypoints without the direct intervention of the drone pilot, and can freely move the gimbal and take pictures by setting suitable camera parameters (e.g., focal length). In other words, the application communicates with the remote controller via USB cable which in turn wirelessly communicates with the DJI Matrice 300 drone (and hence with the DJI Zenmuse H20 camera). Although the Android application is mainly developed for autonomous flights (i.e., path planning, taking pictures, and performing HTTP POST requests to the server), it also allows to manually pick previously stored pictures to send them to the server with the same paradigm.

Basically, the Android application works as follows. Initially, it allows the user to specify the waypoints inside the orchard where the drone needs to take the pictures. On each waypoint, the drone stops and takes around 20 adjacent pictures at different spots of the trees making a grid of 4 cells by suitably rotating the drone's gimbal. Then, the taken pictures are sent to the server.




(a) (b)

Figure 5.5.2: The Android application.

## 5.6 Conclusion

In this chapter, we undertook the study of a system with the ultimate goal of automating the HH pest scouting in orchards by leveraging RGB cameras, drones, and ML algorithms. We initially focused on constructing a suitable dataset of images featuring the HH, and enhancing its quality through a preliminary screening process. Subsequently, adopting the YOLO framework, we trained and evaluated several ML models on the RAND split and employing diverse metrics to comprehensively evaluate their performance. We also assessed the potential impact of blurriness on HH detection, training the aforementioned models on both the BEST and the WORST splits. Our methodology exhibited significant improvements over the results reported in [1] for two main reasons. Firstly, we harnessed on-site images for training, and, secondly, we introduced a novel pre-processing procedure based on a blurriness metric, which further strengthens the effectiveness of the computer vision algorithms. Our results are highly satisfactory and underscore the critical significance of meticulous dataset construction, model training, and image analysis in the successful implementation of ML for HH recognition. The main limitations encountered with our models can be addressed by introducing a detection system that runs on patches instead of entire images in order to improve the attention on small specimens, and to reduce the probability

of suppressing good detections. Furthermore, in order to further improve the precision rate, we plan to plug on top of the YOLO head an additional classification network which will parse the returned detections by distinguishing between stink bug or no stink bug, namely discern between stink bug or tree flaws and shadows.

Further research and developments are required to complete a fully autonomous orchard monitoring, which can be extended to other invasive and emergent pests. To progress towards this goal, several key areas need attention. For instance, the development of a client-server application that leverages the bug-detectors like those previously described for real-time detection is crucial. This application would enhance the practicality and accessibility of the monitoring system. Additionally, integrating bug-detectors with microclimate weather observations to build an HH prediction model holds immense potential. This integration can provide valuable insights into the pest population dynamics, enabling proactive decision-making and pest management strategies. Continued research efforts are essential to identify novel approaches that can complement the existing methods and contribute to more effective IPMs.

# Appendix

## 5.A Impact of Blurriness

The experiments in Section 5.4.3, where the three splits were arranged based on their blurriness scores, seem to suggest that the presence of out-of-focus regions within an image does not pose an issue for the detection. This is evident as the results of the WORST split are not significantly different from the BEST split, and overall, the RAND split achieves the best performance. So, it is worth to explore the impact of blur by deliberately blurring a certain percentage of photos from a specific split. By following this approach, we aim to disprove that increasing levels of blur would paradoxically enhance the detection performance.


Figure 5.A.1: Example of blur kernel effects.

So, we repeat the training of the three YOLO networks by using transformed training and validation set with increasing percentage of blurred images. Figure 5.A.1 depicts the impact of blur kernel effects. For each previously created split, we intentionally introduce blur to 50% and subsequently 100% of the samples. Figure 5.A.2 displays the training performance of the RAND split with 50% of the images transformed. The first row illustrates the loss functions, namely bounding box loss, classification loss, and objectiveness loss. The second row presents the validation metrics, including $P$, $R$, and $m_{0.5}$.

Overall, the training behavior of the models on transformed images is similar to what has already been observed in Figure 5.4.2. Indeed, as regard to the loss functions, we can observe that all of them converge very close to 0 after a few epochs. The $\mathcal{X}$ model demonstrates a tendency to converge faster compared to the other two networks. Conversely, the $\mathcal{S}$ model is the slowest in terms of convergence. When considering the validation performance, we observe a rapid stabilization of all metrics, although each line exhibits some noise in its trend. Hence, we can conclude that, from a training perspective, the introduction of 50% induced blur does not significantly affect the learning performance of the networks. The behavior remains the same even when more blurred images are introduced. Also the training results of the models trained on 100% blurred images do not display any significant discrepancies. So, we avoid to plot them.

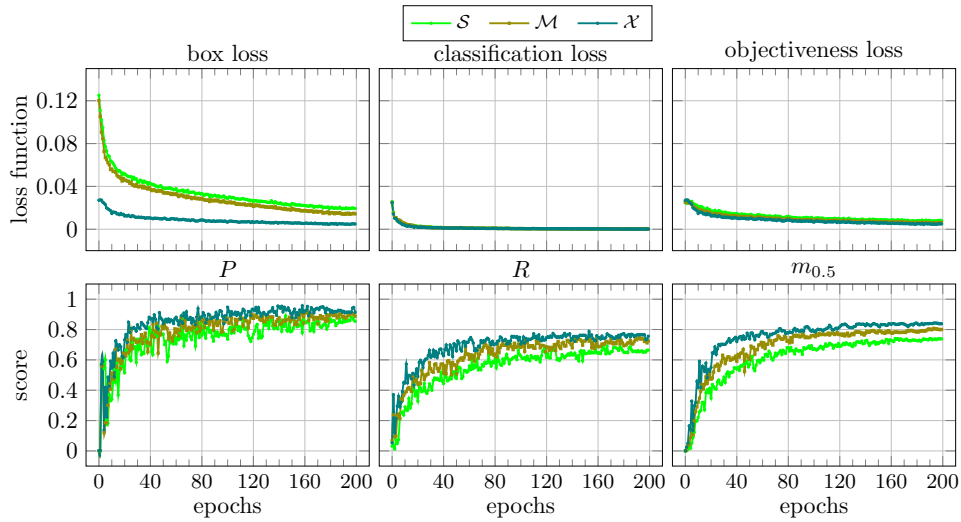



Figure 5.A.2: Training results using RAND split with 50% of blurred images.

Figure 5.A.3 presents the performance evaluation metrics of the three YOLO networks, considering different percentages of forced blur in the training set with 0%, 50%, and 100% of blurred images. Each row of plots in Figure 5.A.3 corresponds to a different group of results, namely HH, NV, and all the classes (ALL), respectively, while each column represents the evaluation metrics. In each plot, the amount of blur imposed on the training set is fixed on the $x$-axis, while the $y$-axis represents the scores obtained by the models.

In principle, we can observe that as the percentage of blur increases, the performance of the networks tends to decrease for each model. When analyzing the behavior of the networks based on their size, we find that $\mathcal{X}$ achieves the highest results, while $\mathcal{S}$ demonstrates the lowest performance, as previously observed. Regarding the precision, all three lines consistently decline as the percentage of blurred images increases, specifically for the HH class. This pattern is also observed in the other rows of the plot. However, we observe the opposite trend for the $\mathcal{S}$ network

and the NV class. We observe a significant increase of 1% in precision when using a training set with 100% blurring. Similarly, when the models are trained on a set with 50% blurred images, we observe a smoother, but still noticeable, increase in precision for $\mathcal{M}$ and $\mathcal{X}$ models. This can be attributed to a degradation in recall, causing the models to become more selective in detecting stink bugs. Consequently, they only predict the clearest targets, resulting in a reduction in false positives and an improvement in precision. The recall is the metric most impacted by blur. For all network sizes, we observe a drop of 2% in recall when using a training set with 100% blurring. This behavior is expected since the ability to recognize unclear objects diminishes. When examining the two mean average precision metrics we observe a consistently decreasing trend, confirming that blurring progressively reduces the recognition capacity of the networks.

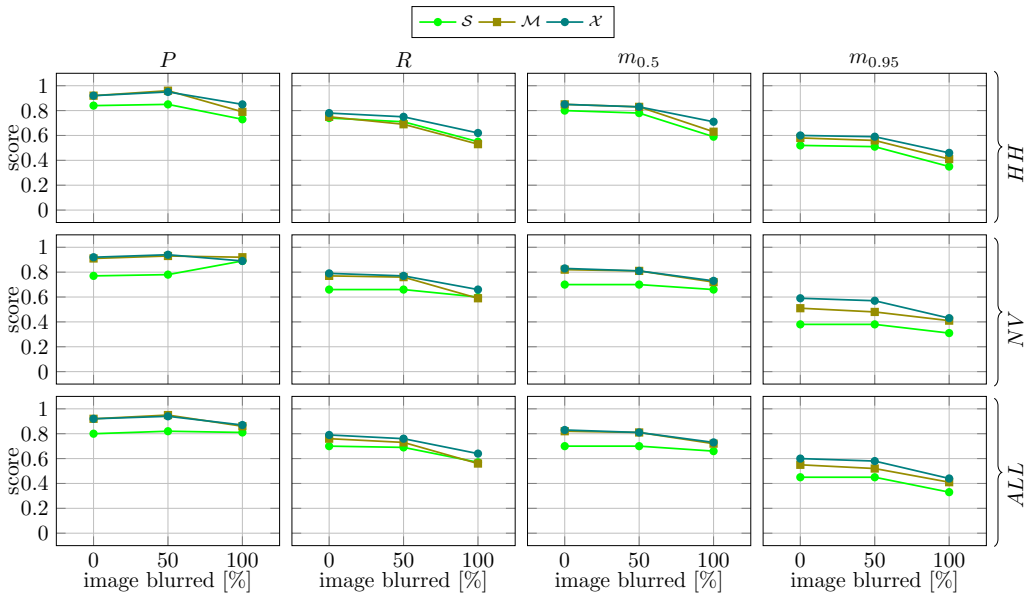



Figure 5.A.3: Evaluation of blur impact on YOLO models using RAND split.

In summary, the results suggest that blurring has a detrimental effect on the recognition capacity of the networks, as indicated by the decreasing trend in all precision metrics.

# Chapter 6

# The Hawk Eye Scan: Halyomorpha Halys Detection Relying on Aerial Tele Photos and YOLOv5

## 6.1 Introduction

During the second season in 2022, significant progress was achieved by focusing on a fully drone-based dataset and leveraging an Android-based application for automatic drone image acquisition. Due to the limitations encountered in the previous data harvesting campaign (Chapter 5), a novel and more reliable protocol was mandatory. These advancements have led to improved outcomes and more efficient data collection for monitoring HH infestations in orchards.

More in details, during the first campaign of the HALY.ID project, the drone acquisition protocol was entirely designed by mimicking the phytosanitary operators, therefore, the drone was driven inside orchard aisles to collect images at several heights of pre-configured waypoints placed in correspondence of the trees. Since the aisles' width is poorly wide, and the size may vary during a mission for the presence of protruding branches, the drone experienced some difficulties in its maneuvers. Precisely, the drone experienced sudden blocks which caused a partial or even worse a complete deadlock. Behind this behavior there are the multiple and simultaneous contribution of the inaccurate GPS tracing, the irregular shape of trees, and the gross sensitivity of the collision avoidance system.

In addition to that, as a consequence of the irregular shape of trees many images taken emerged out-of-focus. Considering that the DJI Zenmuse H20 is a bridge camera built for long range imaging, the resulting DoF in a narrow space is too low, i.e., short distance between the DJI

Zenmuse H20 and the subject. Hence, even the presence of a very tiny object in the foreground can affect the camera focus mechanism, distorting the output image.

The primary objective of the current chapter is defining a new protocol robust against the issues encounter during the first year of the project which can ensure a risk-free flight also in particularly cluttered environments, such as, an orchard. To achieve this, we devise a drone acquisition protocol tailored for our DJI Zenmuse H20 which exploits aerial imaging at 10m. Specifically, the idea is to perform a mission above the orchard at a certain altitude determined by optics law, and which follows a specific path established at the beginning. In this manner, the drone movements are free from obstacle presence, inaccurate GPS mapping, as well as, the quality of the images improves because the altitude of the drone and its position is tuned according to the camera properties. Then, for the purpose of effectively validating the quality of the images collected we devise a thorough analysis on several image features, e.g., blurriness, HH size, emphasizing the usability for Machine Learning tasks. Subsequently, since our last goal is to automatize the entire monitoring process, we propose various computer vision detectors, based on the YOLO framework, trained using image slicing, and three different training sets. Finally, we depict the models performance according both state-of-art and novel metrics, and by varying relevant parameters in the context of object detection, such as, Intersection over Union (IoU), likelihood, highlighting strengths and weaknesses of the networks.

The contributions of this chapter are summarized below.

- We define a novel drone image acquisition protocol for scouting the HH in orchards based on aerial tele-imaging.
- We create the first dataset based on aerial images for HH detection, and completed with both an analysis of the key features, e.g., HH average size and position, and an image quality assessment, i.e., blurriness and brightness evaluation.
- We create several ML models using the YOLO framework on our built dataset in order to detect the HH. When building our ML models, we apply image slicing to avoid information loss, and three distinct approaches for training and validation that use three percentage of negative samples.
- We thoroughly evaluate models performance according to both state-of-art and custom metrics by varying IoU and likelihood thresholds.

The chapter is structured as follows:
Section 6.2 introduces the novel dataset and the computer vision algorithms developed for this

purpose. Section 6.3 examines and discusses the performance of the YOLO-based algorithm on the generated dataset. Finally, Section 6.4 summarizes the main findings and outlines potential future directions. We further motivate the hardware selection, and the selection of the image acquisition parameters for implementing our IPM in 6.A.

## 6.2 Dataset

In this section, we present the dataset that we created. We begin by explaining the composition of the dataset in Section 6.2.1. Then, in Section 6.2.2, we examine the main features that characterize our dataset, such as the size of the bugs in the images. We highlight how to deal with labeling according to this factors.

### 6.2.1 Data Harvesting

In order to autonomously detect the HH on orchards by using drones and computer vision algorithms, a suitable dataset of images containing the HH is required. We relied on the DJI Matrice 300 drone which can be used in combination with the DJI Zenmuse H20 camera, as discussed in the Section 6.A.1. Our objective is to design a mission for the drone that encompasses a predefined path from the initial point to various locations within the orchard. Since the space between orchard's aisles are particularly cramped, the DJI Matrice 300 must fly above the aisles in order to avoid undesired blocks due to the dense foliage and the detect-and-avoid system. Figure 6.2.1 details how the drone flies inside the orchard. According to DJI, the DJI Zenmuse H20 camera needs to be positioned at a sufficient distance from the target, typically at least 8m, in order to capture satisfactory pictures with proper focus. So, since the distance among two consecutive rows is 4m, we decided to fly the drone at 10.5m so that the minimum distance from it and the canopy of the trees is at least 8m.

The images obtained from this process are subsequently utilized to construct a dataset, which serves as the foundation for training the computer vision algorithms. So, we have developed a drone application that enables the creation of missions and facilitates the execution of specific tasks within the orchard. So far, we have mainly studied and then implemented the aisle-missions with drones [9, 10].

Figure 6.2.2 gives an overview [9] of the different stages comprising our Android-based application. We divide the process into three main phases. In the initial phase, *Crop Inspection*, we employ a first-person view to thoroughly survey the orchard whose primary goal is to select a subset of geo-localized points (waypoint) that will be employed for planning the flight. In the

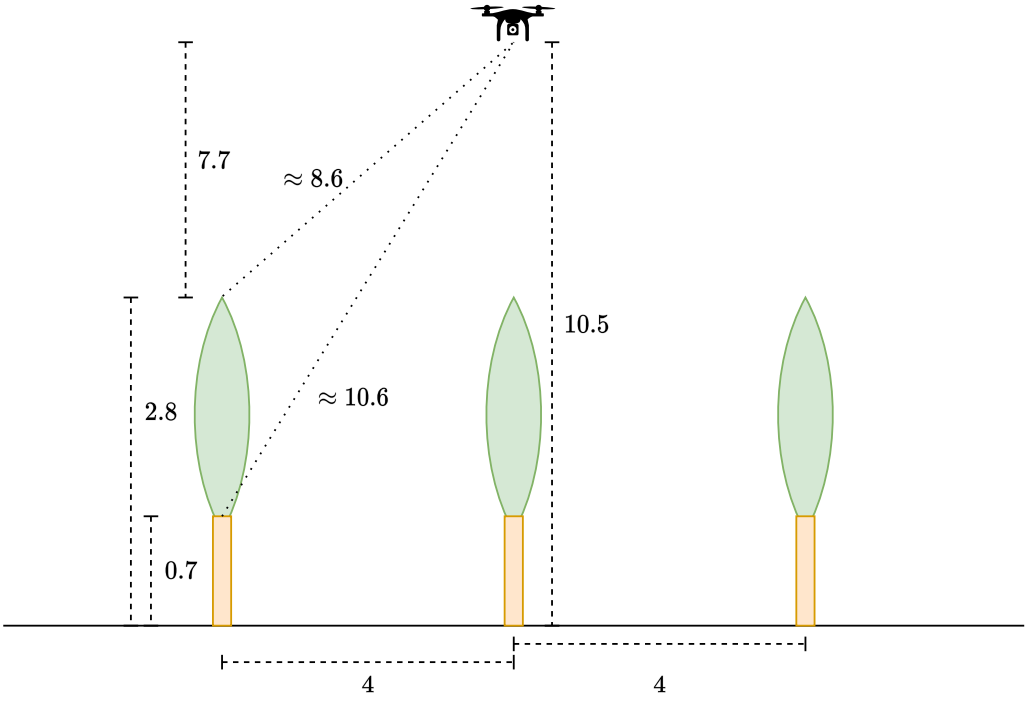



Figure 6.2.1: Illustration on the protocol used by the drone for taking pictures inside the orchard (numbers in meters).

second phase, *Route Planning*, we aim to automate the drone's flight along a designated path passing through the previous waypoints in order to capture images within the orchard. Finally, the third phase, *Bug Scouting*, involves the processing of the captured images whose goal is to detect the presence of bugs though computer vision algorithms. Currently, this phase involves post-processing. However, it is worth noting that the computer vision algorithm can be integrated into the application we are developing to guide the drone.

During the *Crop Inspection* phase, we leverage the expertise of plant operators and entomologists to establish a proper first-person-view route. Our objective is to assign higher priority to surveying the boundaries of the orchard, while giving less priority to the central areas. This decision is based on the operators' knowledge that HH tends to favor locations closer to roads or hedges, making it slightly more challenging to detect HH in the middle of the orchard.

In the *Route Planning* phase, our goal is to compute the drone path including the necessary actions such as capturing pictures at specific locations. To do this, we build a polygonal chain that the drone will follow, using the selected waypoints from the previous phase. At these waypoints, the drone stops to fulfill specific tasks. The drone adjusts its yaw angle and camera angle while setting the focal length to achieve the desired field of view (FoV) given the oblique distance from the tree. Changing the yaw angle, change the oblique distance from the canopy and the drone.


Figure 6.2.2: Overview of the main application core features. It summarizes a three-phase process: *Crop Inspection* involves selecting waypoints for drone flight, *Route Planning* automates flight along waypoints to capture orchard images, and *Bug Scouting* uses computer vision to detect bugs in captured images.

Hence, to maintain a consistent FoV to guarantee the desired image resolution, the focal length needs to be properly set. During picture capturing, the camera benefits from a stable configuration thanks to the presence of a 3-axis gimbal stabilization system. To be more precise, at any waypoint, the drone virtually creates a grid of $4 \times 5$ cells comprising of 20 pictures, each with different focal length in order to keep the same FoV.

The *Bug Scouting* phase focuses on analyzing the captured pictures and examining the results to assess the presence of HH within the orchard. To initiate this phase, we begin by constructing a dataset[1] (detailed in Section 6.2), which is then utilized to train a detection system through computer vision algorithms (results in Section 6.3.1). In our ultimate goal, this detection algorithm will be employed for real-time HH detection. However, at present, our main objective is to conduct extensive testing of the detection algorithm to gain confidence in the feasibility of our overall goal.

### 6.2.2 Data Analysis

The dataset is built using images captured by a drone in an automatic manner from the top of an orchard focusing on a single category of stink bugs. It consists of images depicting the current most harmful stink bug found in Italian orchards, namely the *Halyomorpha halys* (HH). The dataset comprises a total of 402 annotated images, each containing instances of the HH class. However,

[1] Currently, the dataset is kept private as per the HALY.ID consortium agreement. It can be only released on explicit request to HALY.ID's Board. It will be made public for research purposes once the project concludes.

when considering all the images, there are actually 546 distinct occurrences of HH specimen. The two images reported in Figure 6.2.3 offer a glimpse of the complexity of the detection task to face. Indeed, all the images are characterized by a cluttered background made mainly by foliage which camouflages the target, i.e., the HH.

Figure 6.2.3: Example of drone captured images.

However, although the dense and intricated background, the images collected through the proposed protocol present significant similarities (Section 6.2.1). The homogeneous nature of the constructed dataset, derived solely from the drone camera, plays a significant role in shaping characteristics. Indeed, with a single source for image collection, the dataset exhibits standardized image features, such as images shape, bug aspect ratio. These dataset features provide a likely simplified training process and make easier for computer vision algorithms to learn meaningful features from them. Besides the intrinsic features of the images, we observe a consistent similarity between the samples concerning the scenery captured. Concerning this behavior, this is a direct consequence of the stringent protocol built for the data acquisition. Since the drone keeps its distance from the trees constant for the entire mission, and a mission has a limited duration in term of times, i.e., more or less 50 minutes, this allows to retain the environment condition, e.g., light exposition, fixed with almost the same frame subject. It is important to remark that the redundant nature of data collected represents an advantage for the detection algorithms. Specifically, by providing every time a frame with mostly the same background and image conditions will allow to improve their detection performance, even if the models have not generalized well their knowledge during training.

In the following, we are going to survey the dataset from different perspectives in order to understand its quality, and peculiarities. We start our analysis by examining both the size and the the position distributions of our target, i.e., the HH. Figure 6.2.4 reports the bounding boxes

size, and position distribution of the annotated HH, respectively. Specifically, for each bounding box, the left plot shows on the $x$-axis ($y$-axis) its width (height) expressed in pixels (px), whereas the right plot shows its relative coordinates, i.e., $x$ and $y$ components of bounding boxes' center, defined in percentage with respect to the image size.

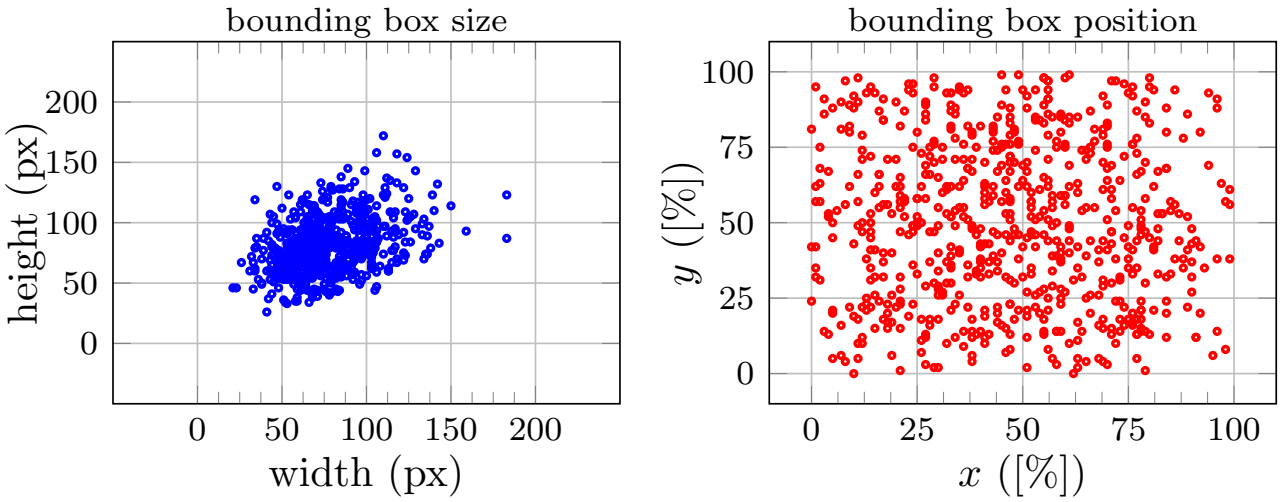



Figure 6.2.4: Bounding box size (left), position (right) distribution.

In principle, we can observe that bounding boxes are small-sized, and with limited variations (Figure 6.2.4, left). Indeed, their size varies between a minimum of 33 $\times$ 45px up to a maximum of 183 $\times$ 132px. The perception of the small-sized bounding boxes becomes stronger when their size is compared with the entire image size, i.e., 5184 $\times$ 3888px. Notice that labeling process was performed by an human operator using the well-known open source cloud service `makesense.ai` [153]. Moreover, paws and antennae were included in the annotations, thus the quantity of pixels attributable to the HH diminishes further. In addition, as a result of the constrained acquisition process, the variability in the bounding box sizes is minimized. Indeed, by keeping the same distance from trees, the specimen photographed in the same pose will be almost same sized. On the other hand, specimen captured with different poses will experience variation in their size. Notice that the majority of the HH annotated in the dataset are captured with a back perspective , namely a dorsal perspective of the bug. This behavior is highlighted by the clustered nature of the plotted points. Only a few points lay out of the cluster center supporting our claim.

Concerning the relative position of the bounding boxes, specifically the center of the bounding boxes, a huge variability is experienced (Figure 6.2.4, right). This is due to the fact that the position of the bugs cannot be established a priori during the flight. In fact, bugs may change several times their position inside the orchard seeking warm temperature, or food. Moreover, the drone has a pre-planned route for taking pictures, and therefore the bugs can be anywhere. For

this reason, the points scattered in the Figure 6.2.4 (right) span practically throughout the entire plot surface.

As aforementioned, the built dataset exhibits limited variation as it is derived from a single source, i.e., the drone. This limited variation results may introduce bias in computer vision algorithms, as indicated in [1]. Moreover, considering that the acquisition process can be affected by various factors, such as failures in the auto-focus protocol and drone flickering during image capturing, we decide to validate the images collected using well-known no-reference estimators from the literature. Specifically, we utilized estimators for both blurriness [145] and perceived brightness [146].

An insight on the perceived blurriness, and brightness is depicted in Figure 6.2.5. In particular, the plot on the left shows the blurriness levels of the collected images, wheres the right one reports the brightness achieved. In both the plots, on the $x$-axis are displayed the images, while on the $y$-axis the metric scores. The blurriness score falls in the range $[0, 1]$, where 0 indicates a totally blurred image, and 1 represents an extremely sharp one. Similarly, the brightness metric is a score ranging from 0 to 1, where 0 states absent lights, and 1 denotes dazzling light. Consequently, images with favorable metrics should have a blurriness score close to 1 and a brightness score around 0.5.

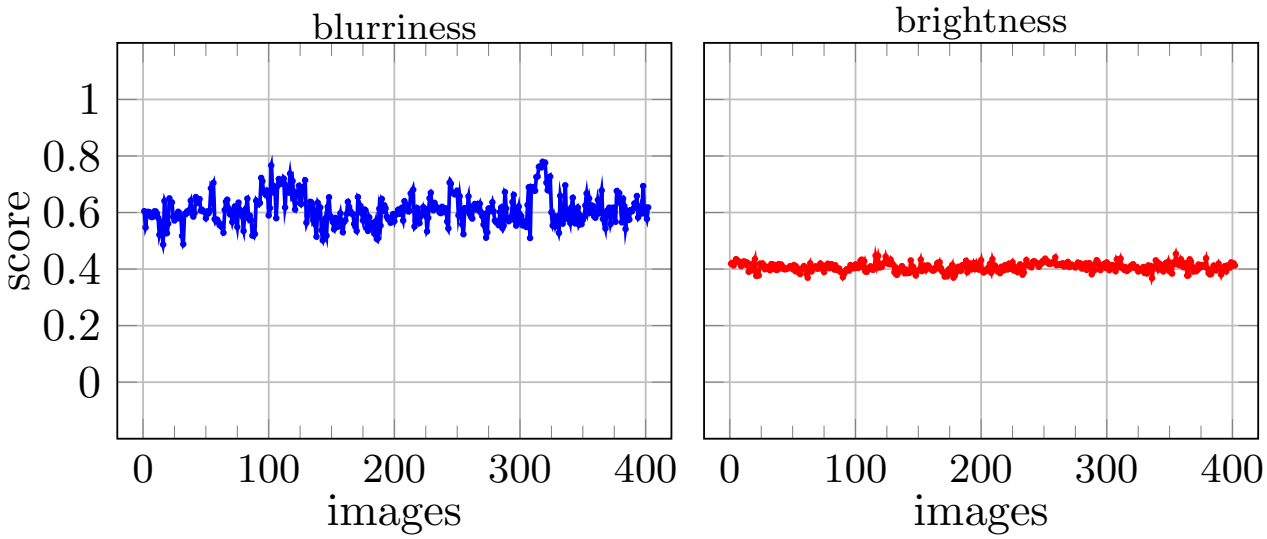



Figure 6.2.5: Image blurriness (left), and brightness (right) evaluation.

From an analysis of blurriness levels (Figure 6.2.5, left), we can notice that all the images achieve satisfactory results, with a stable score around to 0.6. This results were quite predictable because the drone acquisition protocol was strictly configured taking into account optics law, and exploiting also RTK and gimbal stabilization. Indeed, both the RTK and the gimbal stabilization

avoid undesired out-of-focus during unexpected shacking, as well as camera settings settled according to the well-known *hyperfocal focus* technique allows obtaining the entire scene captured sharp. Notice that, the small variations shown is a directed consequence of the focus technique applied. In detail, The hyperfocal distance is the distance between the camera and the optimal point of focus, and is different for every focal length and aperture setting you use, and when a lens is set at the hyperfocal distance, depth of field extends from half this distance to infinity. Therefore, blurred parts occur every time one or more objects stand before the hyperfocal distance. Since trees may have irregular shapes, and the DJI Zenmuse H20 has a limited DoF due to its bridge camera nature, not pruned branches or simply branches moved by the wind may show up closer than expected causing blurred areas in the taken images.

A very similar behavior is shown for brightness (Figure 6.2.5, right). However, in this case differently from the blurriness, since the brightness depends mostly on the environmental light condition, the trend depicted in the plot is almost entirely constant. Indeed, we observe a quite stable trend settled to 0.4 very close to the optimal value, i.e., 0.5. This is due to the scarce variability of light experienced during a mission. Considering that a drone flight is at most 50 minutes, the light levels may not vary consistently in this very short period of time. As directed consequence camera settings settled at the beginning remain optimal for the almost the total of the mission.

To summarize, the dataset demonstrates a satisfactory quality on average as regards blurriness and brightness. The dataset's metrics confirm the effectiveness of the parameters established using both the DJI Matrice 300 and its camera, the DJI Zenmuse H20, and the choices made for the navigation protocol. Additionally, despite the constrained acquisition protocol, the dataset ensures a notable variance in bug position and pose, resulting in a suitable set of samples for training our models.

## 6.3 Evaluation of the Localization and Detection of the Bug

In this section, we detail and discuss the performance evaluation of the HH detection. We revise the main technology adopted for solve our task in Section 6.3.1. We give an overview of the adopted metrics for the comparison in Section 6.3.2. We then thoroughly compare and analyze the training and validation results in Section 6.3.3, and the testing results in Section 6.3.4.

### 6.3.1 Algorithms and Tools

In this section, we provide a concise overview of both the YOLO algorithm and the widely adopted technique known as “Transfer Learning” (TL) that we employed in our approach to detect stink bugs.

The YOLO algorithm, the latest iteration of the YOLO (You Only Look Once) series introduced initially by [154] and more recently enhanced by [143], serves as the core of our experimentation. This deep learning-based architecture, built upon the PyTorch framework, plays a pivotal role in our research. What sets this family of algorithms apart is their unique approach to object detection, reframing it as a regression problem rather than a classification task. This is achieved by spatially separating bounding boxes and assigning probabilities to each detected image using a single Convolutional Neural Network (CNN). One of the standout features of YOLO is its efficiency—it offers a lightweight and speedy solution, demanding considerably less computational resources compared to other contemporary state-of-the-art models, all while maintaining competitive performance [143]. Notably, it boasts the capability to process images at a remarkable rate of 45 frames per second, making it particularly well-suited for real-time bug detection, even on edge-computing devices potentially embedded within the drone itself.

Given our constraints of a limited image dataset and computational power, we made the strategic decision to approach the training phase either from scratch or by adopting the TL paradigm [155]. TL is a valuable technique in computer vision that allows us to leverage aspects of a pre-trained computer vision model that has been extensively trained on a vast repository of images. We can then apply this knowledge to a new model, even when only a limited number of images are available for the new task at hand. This process involves selecting and transferring relevant components from the existing machine learning model to address a novel, potentially related problem.

Central to Transfer Learning is the concept of generalization—only knowledge that can be applied effectively to another model in diverse scenarios or conditions is transferred. This flexibility allows models developed through TL to adapt to changing conditions and varying datasets. Unlike models created from scratch, which are tightly bound to the specifics of their training data, TL-based models are more versatile and can be harnessed across a range of different scenarios and datasets. This adaptability and the ability to extract valuable insights from a pre-trained model are pivotal in our quest to identify stink bugs efficiently and effectively.

### 6.3.2 Adopted Metrics

Before delving into the training and test details, let clarify some essential definitions. As previously mentioned, our detection algorithm is YOLO, and to assess its accuracy in detecting stink bugs, we employ the same metrics previously introduced in Section 5.4.1. Hence, we will evaluate the trained networks using the precision score ($P$), the recall score ($R$), the mean average precision in its two most common forms, namely the m$AP_{0.5}$ ($m_{0.5}$) [147], and m$AP_{0.5:0.95}$ ($m_{0.95}$) [148], respectively. However, considering the complexity of task we introduce an additional version of the mean average precision. In detail, we introduce a novel mean average precision, the $m_{0.25}$, which is a m$AP$ computed similarly to $m_{0.95}$ but starting from an IoU equal to 0.25. Notice that the reasoning behind this metric is strictly tied to the complexity of the recognition. Indeed, since the size of the HH, namely our target, is limited, the $m_{0.95}$ may have a lack of expressiveness. Moreover, differently from other kind of detection problem, here is more relevant to identify all the HH individuals instead of meticulously contour the insect. Hence, we decided to reduce the IoU threshold for highlighting in a clearer way the intrinsic recognition ability of our models.

In the upcoming sections, we will elaborate on how we trained our neural networks using the YOLO models.

### 6.3.3 Configuration of the Object Detectors

In order to train our models we partition the dataset in the following way, 70% is allocated for training, 20% for validation, and 10% for testing. Hence, the original 402 images are grouped in 282 images for training, 80 for validation, and 40 for testing.

Given that the HH area occupies only a minuscule fraction of the entire image, it becomes evident that directly inputting the raw image into the neural network yields unpromising results. Moreover, as emphasized by the authors in [19], the primary cause of performance degradation is the loss of information resulting from image scaling. Indeed, neural networks grapple with high-dimensional data, making their training on large and intricate datasets computationally demanding. Nevertheless, since the image is $5184 \times 3888$px, shrink it to $640 \times 640$px makes practically invisible tiny objects as the HH. To mitigate the risk of losing crucial features due to image scaling, we have chosen to implement a slicing procedure. Therefore, we deal with high resolution images by cropping them in equally sized patches. This approach enables us to reduce the training complexity and put network's attention on specific sub-portions of the image. In view of this, we decide to extract from the images relevant patches of size $640 \times 640$px. Notice that the chosen patch size represents a good trade off between prediction time for the entire image,

and network performance. Indeed, 640 × 640px represents the input setting optimized by YOLO developers, and the size settled for the network weights pre-trained on MS-COCO dataset [148].

Concerning the extraction procedure, we decide to rely on a randomized approach. Basically, the idea behind this procedure is randomly extract sub-portions of size 640 × 640px until the desired number of samples is reached. In detail, we extract randomly patches from original images looking for HH instances, in case surround one of them the relative annotation file is created according to the relative position of the HH inside the patch. By this procedure we obtain 8,000 patches for the training, and 800 for the validation set. Note that, each patch contains exactly one HH. The latter decision was driven by the Figure 6.3.1.

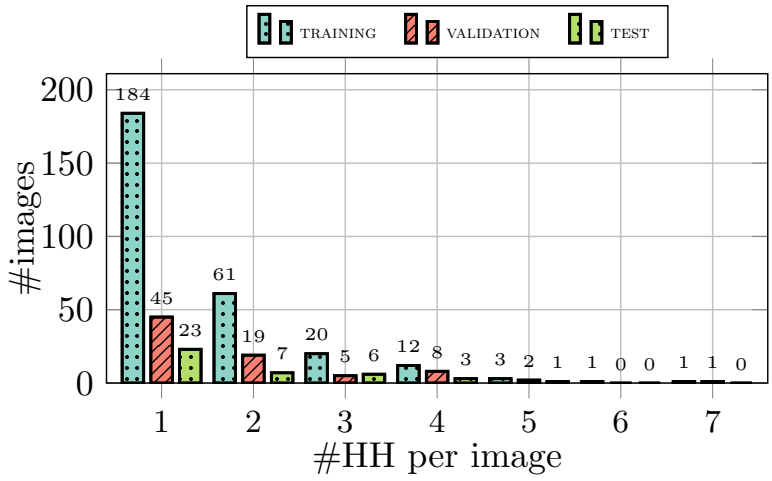



Figure 6.3.1: Distribution of the number of HH per image.

Figure 6.3.1 provides a good insight into the number of HH occurrences per image. The distribution is aggregated according to the split applied. The bar plot shows that the occurrence of HH is relatively rare in our dataset, with most images containing only one instance. In principle, the more HH in an image, the rarer the occurrence of such an image. Moreover, we notice that the percentage of images aggregated per number of HH inside remains stable with limited variations in all three sets. The analysis made emphasizes the predominance of background pixels against HH ones. Hence, it is reasonable to include exactly one HH per patch.

During the extraction procedure we notice that a lot of empty patches are cropped before a positive patch, i.e., a patch with at least one HH inside. Such phenomena happens due to the fact that our targets, i.e., the HH, are spread inside the images very sparsely. Therefore, the low probability of encounter an HH during the slicing procedure or during a drone mission suggests that the models should get used to face low positive samples. Due to this fact neural networks may benefit from training sets with negatives samples, i.e., without any HH inside. For this reason, we create three different sets which differ from the percentage of negative samples. Figure 6.3.2

depicts the composition of the three mentioned training sets. The set $T_0$ contains 100% positive samples both in the training and the validation set, whereas $T_{50}$, and $T_{75}$ are comprised by 50%, and 75% of negative samples, respectively. Notice that, the strategy for extracting negative samples coincides exactly with the one used for positive samples. The only difference is, in this case we look for patches without any HH annotated inside.

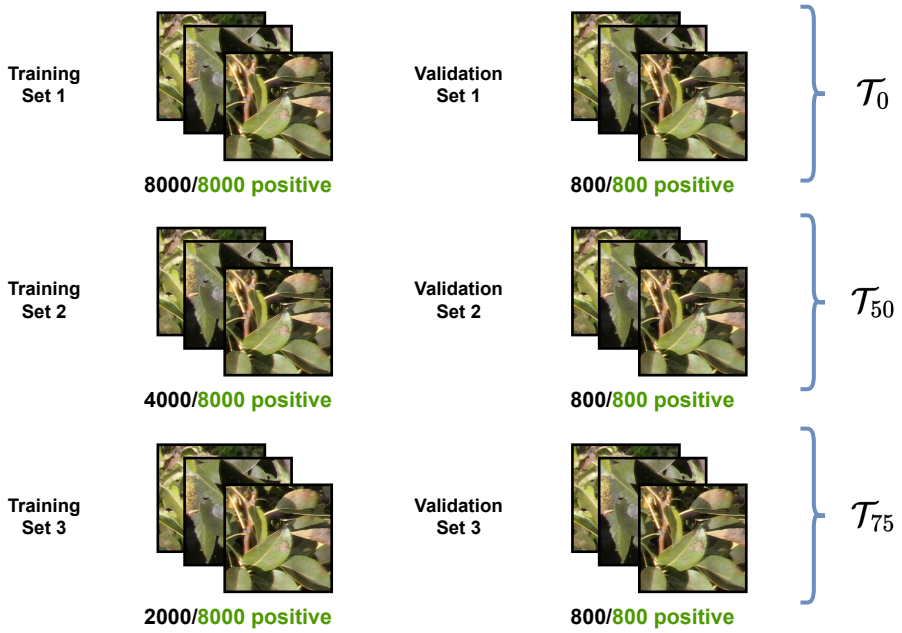



Figure 6.3.2: Training and validation sets composition.

As mentioned, we employ the TL paradigm for training YOLO models. However, since the task is particularly challenging and our hardware not enough powerful we decide to train our models avoiding random weights, i.e., from scratch. The pre-trained weights used were obtained from an intense learning procedure over the well-known MS-COCO dataset [148]. Moreover, we augment the number of training samples by generating three additional images for each training image through random transformations. These transformations involve variation in image hue, saturation, and value (HSV), translation, scaling, flipping, and mosaic techniques, ensuring no image is generated in the same way. Every transformation is computed at each epoch on-the-fly exploiting the well-known Python Library `Albumentation` [156].

The YOLO object detectors are trained using an NVIDIA RTX 3060 with 12GB of VRAM. We rely only on the most powerful YOLO model, i.e., $\mathcal{X}$. Each model underwent training with batches of 32 images for 200 epochs, using learning rate of $10^{-2}$, SGD optimizer [150], and an image size of $640 \times 640$px. The Intersection over Union (IoU) is set to 0.5 to align the predicted bounding boxes with the ground truth better. Notice that, we configure and permit the early stopping for

the training process. Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. The implementation of this method has a twofold impact, avoid overfitting, and simultaneously optimize the resource usage.

Evaluation of experimental results is based on four metrics [157]: precision rate ($P$), recall rate ($R$), mean average precision at IoU equals to 0.5 ($mAP_{0.5}$), and mean average precision with IoU from 0.5 to 0.95 ($mAP_{0.95}$). In Figure 6.3.3, we report the training performance of the resulting models. Each plot displays on the $y$-axis the score, and on the $x$-axis the epochs. The score assumes the meaning of loss function on the first row, where are plotted the values of the most significant loss function, i.e., *box loss*, *classification loss*, and *objectness loss*. On the other hand, through the second row the score represents the value of the metric totalized during the validation, i.e., $P$, $R$, and $mAP_{0.5}$. Before diving in the performance analysis, it is worth to note that, lines plotted may not span over the 200 epochs. This behavior does not represent an issue, and it is a direct consequence of the early-stopping mechanism settled at the beginning.

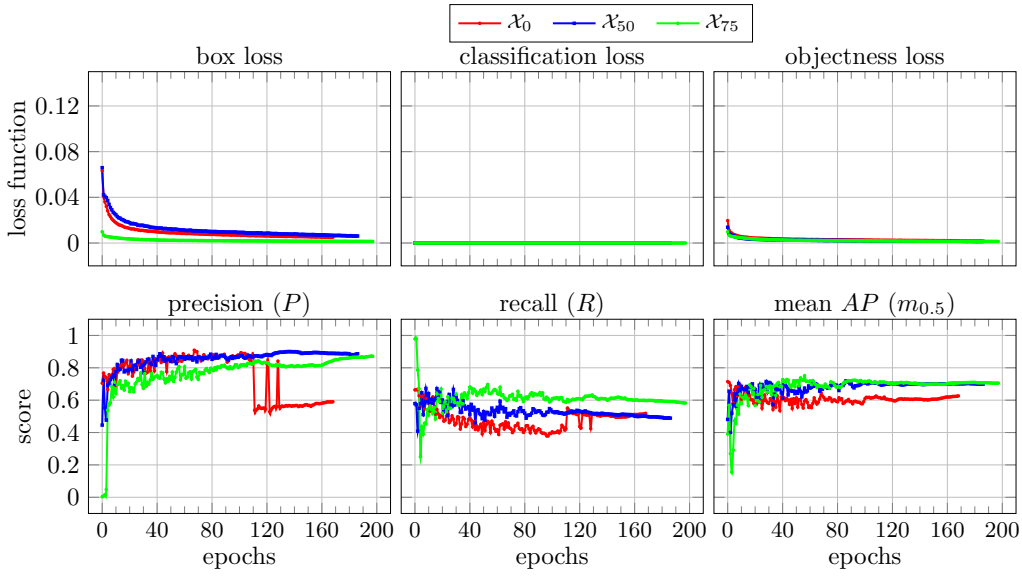



Figure 6.3.3: Training and validation results.

In Figure 6.3.3 (top-left), we observe the *box loss* representing the model's ability to predict bounding box coordinates. All the models reach a low box loss rapidly, although the $\mathcal{X}_{75}$ converges faster. This behavior is probably due to fact that, $\mathcal{X}_{75}$ is the model trained upon the training set with the minimum number of positive samples, i.e., $\mathcal{T}_{75}$. Therefore, this means that is simpler to seek a function that fits all the boxes. However, it is important to remark that after almost 50 epochs every model reaches a value very close to 0. Concerning the *classification loss* (Figure 6.3.3, top-center) the functions trends are constant, i.e., equal to 0. This is a direct consequence that there is not a classification problem in our task. Indeed, we have only bug specimen to recognize,

i.e., the HH. In Figure 6.3.3 (top-right) is reported the the *objectness loss*, which is roughly speaking the confidence that some object exists in a given box, rapidly converge for all the models and all the groups. One can notice that the objectness loss, although it reaches a plateau, remains slightly higher than 0.

The remaining three plots reported in the second row display the metrics achieved by the models during validation phase. In principle, all the models achieve good performance and tend to stabilize after a few epochs. Although they evolve with a similar trend, every model stands out because of some peculiarities. As regards the precision $P$ (Figure 6.3.3, bottom-left), the model $\mathcal{X}_0$ registers immediately the highest value maintaining it up to roughly 100 epochs. After that, $\mathcal{X}_0$ becomes unstable with multiple peak and drops. This behavior is due to the complexity of the detection task, and the noise featured on the HH annotated. In other words, since the annotations in the training set have several aspect ratio, shapes, levels of occlusion, and so on, the learning procedure experienced good results with a more general recognition function which becomes every epoch more complex in order to fit with all samples. This means that once the function reaches a good level of knowledge every alteration may represent a huge drop or improvement for $P$. Differently, $\mathcal{X}_{50}$, and $\mathcal{X}_{75}$ perform with a smoother trend. Indeed, $\mathcal{X}_{50}$ quickly increases the $P$ value, and reaches a plateau at $\approx$ 0.83 after 20 epoches. The same reasoning can be applied for the model $\mathcal{X}_{75}$, although its trend grows slowly after an initial huge jump. It seems that the learning process improves its performance on $P$ by including new images previously not handled. In Figure 6.3.3 (bottom-center) are depicted the $R$ trends observed during the validation step. The behavior of the three models moves with the same reasoning stated for the $P$ aside a few difference. In particular, $\mathcal{X}_0$ achieves the best $R$ score at the beginning, and then, after a rugged descending trend, stabilizes at $\approx$ 0.58 score. Again, this may follow by the fact the $\mathcal{T}_0$ is the set of images with the maximum number of HH instances, thus the most complex to approximate. The model $\mathcal{X}_{50}$ similarly performs better at the very first 30 epochs, following with slight descending bending. The $\mathcal{X}_{50}$ in contrast with $\mathcal{X}_0$ is characterized by a smoother trend. In principle, the observed curves behavior, namely a raising trend at beginning and a subsequent descending trend, is a direct consequence in view of the $P$ performance. Indeed, we can note that the $P$ lines move complementary with respect to $R$ lines. Probably, at the very fist stage of the learning process the algorithms tend to predict with more optimism, at the cost of poorer $P$. On the other hand, increasing the epochs back-propagation takes action to improve the overall performance diminishing slightly models optimism in bounding box proposal. $\mathcal{X}_{75}$ is the model which registers the best $R$ score during the training phase. However, the observations made for the other models still remain valid also for it. Concerning the $mAP_{0.5}$ (Figure 6.3.3, bottom-right),

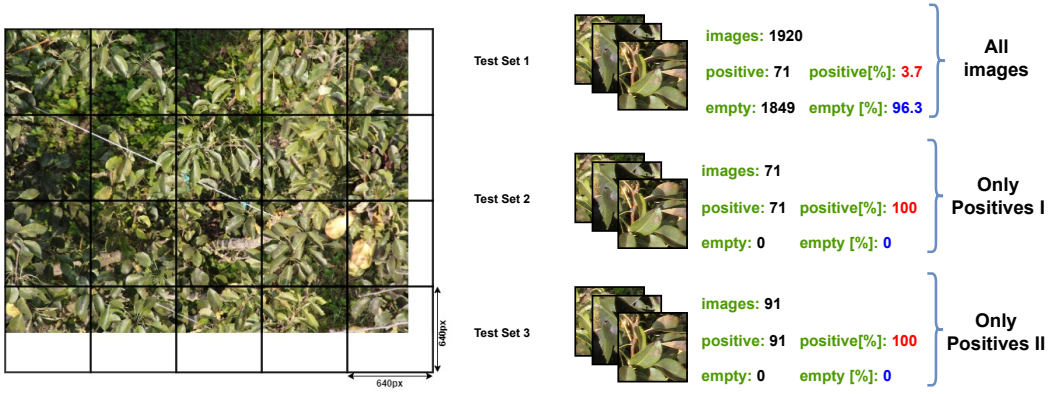
the very last parameter used to understand models performance during training, we can notice that every model has an initial settling period with a consequent stabilization comprised between 0.60 and 0.72 score. Since $mAP_{0.5}$ is a metric to estimate $P$ and $R$ performance under several confidence levels, it is quite clear from lines behavior that all the models are confidence-robust.

Summarizing, we note that all the models obtain satisfactory performance with both the loss functions and metrics. Although some models portend that increasing the number of training epochs will result an increment of metrics performance, we observe that only one metric of the three analyzed has an ascending trend. For this reason, since most of the metrics stabilize their trends reaching good performance within the epoch range, we can state that 200 epochs represent a reasonable trade-off between quality of the solution, and training time. Furthermore, we observe that the models trained on increasingly amount of negative samples, i.e., training set with higher percentage of empty images, reach overall the best results on the metrics with a stable and smooth trend. On the under hand, the models trained on higher amount of positive samples exhibit the "worst" performance, with a notable unstable and rugged trend.

### 6.3.4 Testing Results

For evaluating the detection ability of the trained models we use the remaining 10% of the dataset, i.e., 40 images. In this case, differently from training, and validation we extract the patches from the test set by applying a regular grid of cell $640 \times 640$px. Figure 6.3.4a depicts the procedure applied to slice in patches an image. Since both the side sizes of the collected images are not divisible by 640, the very first step is applying a black padding on both the sides. This ensures that every image can be always partitioned in patches all of the same size. Then, the slicing procedure is applied, and simultaneously the bounding boxes, i.e., the HH instances, encountered are adapted according to the patch size, and their relative position. In this way, employing this extraction procedure a new test set of $1,920$ patches is obtained. As consequence of the slicing procedure, some instances are splitted in two different patches. In detail, of the 71 original instances contained in the input test set, 20 of them were cut due to the slicing. It is worth noting that, the annotations have been made by an expert operator directly on the entire images. As a consequence, any operation performed over the annotation files starts from the entire image and it is strictly related to its original size. Since that slicing procedure may partially include HH into a patch, we apply a thresholding strategy that considers a valid annotation the only one with at least the 50% of the original pixels. In case an HH is cut exactly 50% between two patches, a random selection of only one half is performed. For this reason we decide to create three different sets of annotations in order to understand more deeply the strengths and the limitations

of the models.




(a) (b)

Figure 6.3.4: Patch extraction for test images (a); and insight on the three test configurations (b).

Figure 6.3.4b illustrates the composition of the three test sets. The first set, named *all images*, contains both positive and negative patches. However, since some HH instances fall between two patches, it would be unfair to search for 91 HH instances instead of the original 71. Due to this fact, we retain all the instances which fall inside a patch up to half of their original size. It is important to note that if an HH occurrence is exactly halved between two patches, we randomly select only one of the two portions. Additionally, we retain all the instances that reside entirely in one patch. Therefore, the *all images* test set comprises $1,920$ patches, with 71 of them containing exactly one HH instance, and the remaining $1,849$ being empty. The *all images* test set allows us to measure the models performance in real-life scenario. Next, we create a second test set, named *positive patches I*, which contains the 71 positive patches from the *all images* set. This set is designed to assess the true precision of the models and allows for a comparison of sensitivity to false positives (FP) with and without negative patches. In other words, the *positive patches I* set is a way to estimate the theoretical performance of our models, that is, a measurement in a controlled environment where the amount of pixels without the HH is balanced. Finally, we establish a third set, called *positive patches II*, which contains 91 positive patches. Essentially, this set includes all patches that contain a portion of an HH, regardless of its size. In this case, the purpose of this set is to evaluate the models' ability to detect occluded bug instances.

Table 6.3.1 presents the testing results of the YOLO models trained using TL with pre-trained weights under several combination of IoU and confidence thresholds. In particular, Table 6.3.1 reports the metrics score achieved by the $\mathcal{X}$ model trained on the three training sets, i.e., $\mathcal{T}_0$, $\mathcal{T}_{50}$, and $\mathcal{T}_{75}$. In addition to the metrics discussed in Section 6.3.2, we report also the number of true positives (TP), false positives (FP), and false negatives (FN). Then, to ensure an overall picture

of detector performance we vary two IoU value, i.e., 0.25, and 0.5, in combination with two confidence threshold, i.e., 0.3, and 0.1. Notice that, we report only the most significant results for the sake of ease.

In principle, the results obtained for the set *all images* consistently yield lower scores compared to the other two sets. This is expected, given that the *all images* set is the largest one. When comparing the overall performance between *only positives I* and *only positives II*, we observe that *only positives I* consistently exhibits higher values of $R$ (recall) but consistently underperforms in $P$ (precision) scores. This discrepancy arises from differences in the number of occurrences. The set with more *HH* instances will naturally result in at least the same number of *HH* instances being detected compared to the smaller set, consequently yielding a higher $P$ score. On the other hand, the smaller set enables models to miss fewer instances, leading to a higher $R$ score. Let us observe that in the context of scouting a bug it is better to prefer precision to recall. Namely, it is more acceptable to lose some bugs than raise unjustified alerts.

Concerning the results with respect to the percentage of negative samples in the training set, the best performance is achieved when using $\mathcal{T}_{75}$ as the training set, which has the highest percentage of empty images. This outcome can be attributed to the fact that the majority of the extracted patches are empty, with very few containing a bug. Consequently, an increasing percentage of empty images during training prepares the detector to handle the low presence encountered in the test sets.

As a direct consequence, the models are less proned to propose a large number of bounding boxes, thereby reducing the number of false positives ($F_P$). Moreover, the scarcity of positive samples in the training set encourages the models to better generalize their knowledge, leading to a reduction in the number of false negatives ($F_N$). Therefore, in the opposite way, reducing the number of negative samples in the training set results in decreased overall model performance, as indicated in Table 6.3.1.

In terms of Intersection over Union (IoU), it is evident that the models are heavily dependent on its value from two perspectives. Initially, we observe that metrics are generally higher when the IoU threshold is set to a low value. This is expected, as HH instances are relatively small in comparison to the entire scene, making it easier to achieve true positives ($T_P$) with bounding boxes that intersect minimally with the ground-truth. However, we can also observe that the models are not so IoU-robust, as evidenced by the disparity between $mAP_{0.25}$ and $mAP_{0.95}$, both of which significantly differ from the $P$ score. This suggests that the majority of the predicted bounding boxes exhibit low levels of IoU with the ground-truth.

Regarding the confidence robustness of the models, they generally perform well. Focusing on

| conf | IoU | model | set | $T_P$ | $F_P$ | $F_N$ | P | R | $\mathbf{m}_{.5}$ | $\mathbf{m}_{.25}$ | $\mathbf{m}_{.95}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.3 | 0.5 | $\mathcal{X}_0$ | all patches | 33 | 30 | 38 | 0.52 | 0.47 | 0.43 | 0.28 | 0.2 |
| | | | only positives I | 33 | 5 | 38 | 0.87 | 0.47 | 0.45 | 0.3 | 0.21 |
| | | | only positives II | 35 | 5 | 56 | 0.88 | 0.38 | 0.38 | 0.24 | 0.17 |
| | | $\mathcal{X}_{50}$ | all patches | 34 | 30 | 37 | 0.53 | 0.48 | 0.4 | 0.26 | 0.19 |
| | | | only positives I | 34 | 2 | 37 | 0.94 | 0.48 | 0.47 | 0.31 | 0.22 |
| | | | only positives II | 36 | 2 | 55 | 0.95 | 0.4 | 0.4 | 0.25 | 0.18 |
| | | $\mathcal{X}_{75}$ | all patches | 38 | 33 | 33 | 0.54 | 0.54 | 0.43 | 0.27 | 0.2 |
| | | | only postives I | 38 | 2 | 33 | 0.95 | 0.54 | 0.53 | 0.34 | 0.24 |
| | | | only positives II | 40 | 2 | 51 | 0.95 | 0.44 | 0.43 | 0.27 | 0.19 |
| | 0.25 | $\mathcal{X}_0$ | all patches | 34 | 29 | 37 | 0.54 | 0.48 | 0.44 | 0.28 | 0.2 |
| | | | only positives I | 34 | 4 | 37 | 0.89 | 0.48 | 0.47 | 0.3 | 0.21 |
| | | | only positives II | 36 | 4 | 55 | 0.9 | 0.4 | 0.39 | 0.24 | 0.17 |
| | | $\mathcal{X}_{50}$ | all patches | 34 | 30 | 37 | 0.53 | 0.48 | 0.4 | 0.26 | 0.19 |
| | | | only postives I | 34 | 2 | 37 | 0.94 | 0.48 | 0.47 | 0.31 | 0.22 |
| | | | only positives II | 36 | 2 | 55 | 0.95 | 0.4 | 0.4 | 0.25 | 0.18 |
| | | $\mathcal{X}_{75}$ | all patches | 38 | 33 | 33 | 0.54 | 0.54 | 0.44 | 0.29 | 0.2 |
| | | | only positives I | 38 | 2 | 33 | 0.95 | 0.54 | 0.52 | 0.34 | 0.24 |
| | | | only positives II | 40 | 2 | 51 | 0.95 | 0.44 | 0.43 | 0.27 | 0.2 |
| 0.1 | 0.5 | $\mathcal{X}_0$ | all patches | 35 | 83 | 36 | 0.3 | 0.49 | 0.45 | 0.29 | 0.2 |
| | | | only postives I | 35 | 6 | 36 | 0.85 | 0.49 | 0.48 | 0.31 | 0.22 |
| | | | only positives II | 37 | 6 | 54 | 0.86 | 0.41 | 0.39 | 0.25 | 0.18 |
| | | $\mathcal{X}_{50}$ | all patches | 39 | 76 | 32 | 0.34 | 0.55 | 0.43 | 0.28 | 0.2 |
| | | | only postives I | 39 | 4 | 32 | 0.9 | 0.55 | 0.54 | 0.35 | 0.25 |
| | | | only positives II | 42 | 4 | 49 | 0.91 | 0.46 | 0.45 | 0.28 | 0.2 |
| | | $\mathcal{X}_{75}$ | all patches | 48 | 151 | 23 | 0.24 | 0.68 | 0.49 | 0.32 | 0.22 |
| | | | only positives I | 48 | 6 | 23 | 0.89 | 0.68 | 0.65 | 0.43 | 0.3 |
| | | | only positives II | 50 | 6 | 41 | 0.89 | 0.54 | 0.53 | 0.34 | 0.24 |
| | 0.25 | $\mathcal{X}_0$ | all patches | 36 | 82 | 35 | 0.31 | 0.51 | 0.46 | 0.29 | 0.2 |
| | | | only positives I | 36 | 5 | 35 | 0.88 | 0.51 | 0.49 | 0.31 | 0.22 |
| | | | only positives II | 38 | 5 | 53 | 0.88 | 0.42 | 0.41 | 0.25 | 0.18 |
| | | $\mathcal{X}_{50}$ | all patches | 39 | 76 | 32 | 0.34 | 0.55 | 0.43 | 0.28 | 0.2 |
| | | | only positives I | 39 | 5 | 32 | 0.89 | 0.55 | 0.54 | 0.35 | 0.25 |
| | | | only positives II | 42 | 5 | 49 | 0.91 | 0.46 | 0.45 | 0.28 | 0.2 |
| | | $\mathcal{X}_{75}$ | all patches | 49 | 150 | 22 | 0.25 | 0.69 | 0.5 | 0.32 | 0.22 |
| | | | only positives I | 49 | 5 | 22 | 0.91 | 0.69 | 0.67 | 0.42 | 0.3 |
| | | | only positives II | 51 | 5 | 40 | 0.91 | 0.56 | 0.54 | 0.34 | 0.24 |

Table 6.3.1: Results $\mathcal{X}$ models on test sets.

$mAP_{0.5}$, when examining the *all images* set, its value is nearly equal to $P$. This suggests that the detector predicts bugs with high confidence levels. However, the $P$ achieved by the other two test sets is roughly twice the values of $mAP_{0.5}$, indicating that the slightly more than half of the detection possess a high level of confidence. Although, the latter consideration, the performance related to the confidence are more than satisfactory if considered in view of the task's complexity.

In conclusion, a significant improvement in the results has been achieved when compared to the initial approach outlined in [19]. Specifically, employing a slicing mechanism for both training and testing, along with the introduction of negative samples, has enabled the models to outperform the state-of-the-art results on the dataset presented in [19]. This achievement can be attributed primarily to the slicing procedure, which allows the models to analyze images at different scales, eliminating the information loss issue highlighted in [19]. Moreover, although the observed performance are a little bit the ones achieved in Chapter 5 it is important to point out that in this latter case the image acquisition must be at supervised, or even worse completely manual. For this reason, the preliminary results presented in this section stand as a promising starting point for automated IPM.

## 6.4 Conclusion

In this chapter, we presented methodologies employed in a drone-based application designed to automate HH pest scouting in orchards. We created a novel dataset of images containing HH specimens by using the DJI Zenmuse H20 and the DJI Matrice 300. We devised an Android-based application in order to automatize the collection of images in orchards, incorporating a novel custom navigation protocol that involves capturing images from a top perspective. Unlike previous efforts [1, 140], the developed navigation protocol effectively overcomes the challenges associated with drone size, which previously hindered safe movements through orchard aisles, as well as ensuring high-quality images. The experiments conducted with the YOLO algorithm for HH detection using a dataset comprised exclusively of DJI Matrice 300 images showcased promising results, demonstrating the feasibility of automated recognition of the HH in challenging scenarios.

As future works, current results suggest that elaborating images with attention on portions, thus computing detection on patches, could represent a potential strategy to enhance the performance even further. So, we plan to investigate the impact of the patch size on the detection performance. Moreover, it is worth to compare the performance of our models with further single-shot detectors, in turn the most recent releases of the YOLO framework, i.e., YOLOv7, and YOLOv8,

and two-shot detectors, such as Faster-RCNN [158], as well as, make a try with the novel detection transformer, or zero-shot detectors [159]. Moreover, since both the DJI Zenmuse H20 and DJI Matrice 300 emerge as promising solutions for stink bug scouting, we would like to consider the possibility of integrating multispectral sensors cameras.

In conclusion, our research runs as a catalyst for computer scientists to investigate technology transfer within the realm of sustainable agriculture. By continuing to explore and develop innovative solutions, we can collectively contribute to the promotion of sustainable practices and resilience against pest-related challenges.

# Appendix

## 6.A Hardware Selection

In this Appendix, we discuss the essential hardware requirements of a vision chip (Section 6.A.1), and the selection of a drone capable of carrying the camera (Section 6.A.2) and able to follow a certain navigation protocol tunable according to the orchard shape and requirements.

### 6.A.1 Vision Chip Selection

In the HALY.ID project we aim to monitor and hence to capture stink bugs of size 1cm $\times$ 1.4cm on pictures with all their peculiar features visible. HH belongs to the Pentatomidae family, and for reliable monitoring it is necessary to distinguish it from other specimens of the same family that can occasionally be found in orchards [160], such as the green bug *Nezara viridula* (NV), and *Raphigaster nebulosa* (RN), which due to its gray-marbled coloring can easily be confused with HH (see Figure 6.A.1, top). So far, only NV and HH specimens have been found in our monitored orchard. Figure 6.A.1 (bottom) reports the main characteristics of HH. It has striped bands on the antennae, the head is rectangular, there are five clearer dots, aligned and well marked, both on the pronotum disc and at the base of the scutellum (the triangular shaped structure between the wings), and a typical pattern of the connexivum (the flattened lateral edge of the abdomen) with checkered medial triangular shaped spots.

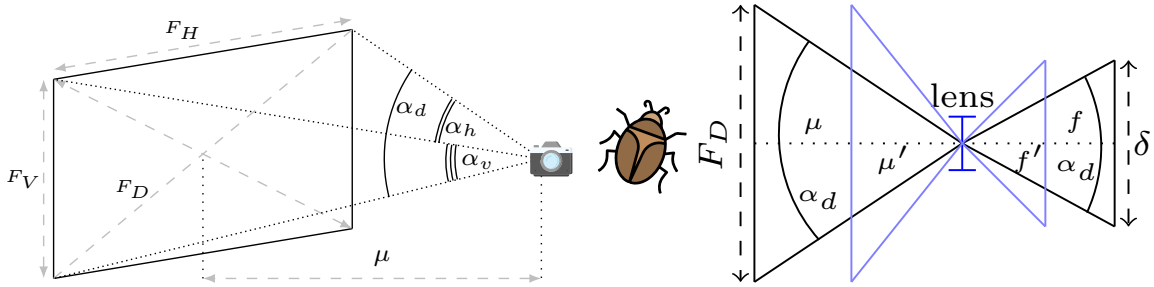
It is evident that a suitable camera/drone combination is detrimental in order to scout the presence of these bugs. Therefore, we need to evaluate some chip vision variables and parameters with respect to the Optics theory. According to Optics theory, the minimum distance between two distinguishable points is called the *resolution distance* $\sigma$. It can be defined as the size of the view that is impressed in a single pixel. In order to identify the small characteristics of HH (see Figure 6.A.1, bottom), a picture with $\sigma = 0.2$mm is desirable so as each feature falls in one or more pixels and hence it is distinguishable. However, if the resolution is reduced to $\sigma = 0.2$mm, each characteristic would be spread across multiple pixels, making it still visible.

Figure 6.A.1: Three Pentatomidae stink bugs (top): *Halyomorpha halys* (HH, left), *Nezara viridula* (NV, center), and *Raphigaster nebulosa* (RN, right, never observed in the monitored orchard); HH and its distinctive features (bottom).

The *Field of View* (FoV) is the size of the view that includes the subject that will be reproduced on the chip. There are three types of field of views (FoVs): Horizontal, Vertical, and Diagonal, i.e., HFoV, VFoV, and DFoV (briefly $F_D$), respectively. The DFoV summarizes both the HFoV and the VFoV, and it is used to simplify the image analysis. Similarly, the size of the image sensor chip is typically described using three dimensions: the width ($n_h$), the height ($n_v$), and the diagonal ($n_d$). The solid angle at the lens, opposite to the FoV, is called *Angle of View* (AoV), and for the analysis we use the Diagonal AoV (briefly DAoV) $\alpha_d$. To guarantee the $\sigma$ distance resolution, we aim to acquire photos with $F_D = \sigma n_d$, which is the largest diagonal view able to distinguish features of size $\sigma$ with a chip of size $n_d$.

When the HALY.ID project began in February 2021, the vision chips commonly embedded in medium/small commercial drones had a chip resolution (i.e., number of pixels) of 12MP. Such chips, with $n_d = 5000$, were mounted on two drones, i.e., DJI DJI Phantom 4, a medium range drone, whose weight is 1830g, with diameter of 35cm (excluded propellers), and DJI DJI Mini 2, a mini lightweight drone ($< 250$g) with diameter of 26cm (including propellers). These drones have an embedded camera with a predefined DAoV of $\alpha_d = 94^\circ$ for the DJI Phantom 4, and $\alpha_d = 83^\circ$ for the DJI Mini 2. Fixed $\alpha_d$, the $F_D$ that satisfies the desired resolution $\sigma$ is achieved at the distance $\mu = \frac{F_D}{2 \cdot \tan(\alpha_d/2)} = \frac{\sigma n_d}{2 \cdot \tan(\alpha_d/2)}$. So, the DJI Phantom 4 and DJI Mini 2 can achieve the desired $F_D = 0.2 \cdot n_d = 1000$mm at a distance $\mu \leqslant 466$mm and $\mu \leqslant 565$mm, respectively, from the view subject (the bug), i.e., both $\approx 0.5$m. As the distances are below the range of the drones' obstacle avoidance systems (usually 1m), they will prevent them from approaching the subject too closely, which may result in the acquired photos not providing the desired resolution.

Since the aforementioned embedded cameras are not suitable for our goal, we then consider




(a) (b)

Figure 6.A.2: FoVs and AoVs from a camera (a); and two configurations (one black, one blue) with the same $F_D$ (b).

the CMOS chip of the DJI DJI Zenmuse H20 (Figure 6.A.3, right) camera. This is an RGB camera mounted as an external payload on the DJI Matrice 300 (Figure 6.A.3, left). The chip has effective 20MP, precisely $n_v = 3888$, $n_h = 5188$, and $n_d = 6483$ pixels, and height $v = 5.70$mm, width $h = 7.60$mm, and diagonal $\delta = 9.60$mm. The DJI Zenmuse H20 camera is a zoom camera whose focal length varies from 6.83 to 119.94mm, and its DAoV $\alpha_d$ varies from 4° to 66.6°. So, the largest diagonal view that guarantees $\sigma = 0.2$mm is $F_D = 0.2 \cdot 6483 = 1296$mm. Varying $\alpha_d$, $F_D$ can be achieved at different distances from the subject. For example, in Figure 6.A.2b the blue and black triangles in front of the lens represent two images with the same $F_D$, but different distances ($\mu'$ and $\mu$). Fixed $\alpha_d$, $F_D$ that satisfies the desired resolution $\sigma$ is achieved at the distance $\mu = \frac{F_D}{2 \cdot \tan(\alpha_d/2)} = \frac{\sigma n_d}{2 \cdot \tan(\alpha_d/2)}$. Since the DJI Zenmuse H20 is a zoom camera, fixed the desired resolution $\sigma = 0.2$, for each $\alpha_d$ between 4° and 66.6°, the desired $F_D = 1296$mm can be achieved selecting a suitable distance between 18.5m and 0.98m.

So, we eventually adopted the DJI Zenmuse H20 camera which achieves the desired $\sigma$ in a large range of distances.

Figure 6.A.3: The DJI Matrice 300 drone (left) and the DJI Zenmuse H20 camera (right).

### 6.A.2 Drone Selection

From the above discussion about the camera, we selected the DJI Matrice 300 drone [10] (6.3kg) which is able to carry the DJI Zenmuse H20 ($\approx$ 0.7kg) as a payload, and utilizes it. The DJI Matrice 300 has a collision avoidance system that prevents it from approaching any target within 1m. In addition, with open propellers, it has a diameter of $\approx$ 1m. Therefore, the minimum width of a single aisle inside an orchard should be $\geqslant$ 3m, considering the size of the drone as well as the collision avoidance system. Moreover, the DJI Matrice 300 is equipped with real-time kinematic positioning (RTK), which allows for extremely accurate global positions from satellites, in the order of 1.5cm of displacement. While this level of accuracy would be beneficial if more flight space were available, the distance between trees on adjacent rows is typically 3.5–4m, and the available room can sometimes decrease below the required 3m, especially when a long time has passed since pruning. So, often it is required to take pictures from different angles instead of in front of the trees.

# Conclusions

Nowadays, we are living a turning point for the adoption of a plethora of technologies which are revolutionizing our daily life. Among these innovations, drones are rapidly imposing in a lot of common applications, especially in last-mile delivery, and smart agriculture. Indeed, they are reshaping the current logistics methods by overcoming geographical constraints, shortening delivery times, and lowering carbon footprints. As well as, the introduction of drones in smart agriculture applications redefines the concept of performance, posing a new efficiency standard with respect to human.oriented solutions.

Along this dissertation we have explored several problem that arise in the dynamic intersection between UAVs and human intensive activities. Starting from Part I, we have delved into the drone last-mile delivery scenario, proving the efficiency and effectiveness of the proposed methods towards the optimization of drone trajectories into a windy environment by solving the MDP. Moreover, we have addressed, by proposing several algorithms, the optimization of the delivery scheduling assuming the cooperation between a fleet of drones and a truck.

In the Part II we have explored novel routing problems that arise in the context of smart agriculture, proposing for them tailored solutions.

Finally, in the Part III we have investigated the prominent field of Pest monitoring using the synergic combination of drones and artificial intelligence. Specifically, our efforts center on the development of a Integrated Pest Monitoring (IPM) specialized for a particularly invasive and harmful insect, the Halyomorpha Halys (HH), colloquially known as “brown marmorated stink bug” (BMSB).

As future works, the very first extension could be towards the integration of the monitoring algorithms proposed in Part I as navigation aware methods for the Android application devised in Part II. Another interesting innovation could be the porting of computer vision algorithms on an embedded system board, e.g., a NVIDIA Jetson, to perform real-time detection in the field hidebound by an internet connection requirements. Finally, it worth to extend the achievements obtained in Part I into the framework of a dynamic environments where wind conditions, and user demands may vary during a mission.

# Publications

## Journal Articles

1. Francesco Betti Sorbelli, Federico Corò, Punyasha Chatterjee, Sajjad Ghobadi, **Lorenzo Palazzetti**, and Cristina M. Pinotti, "*A Novel Graph-Based Multi-Layer Framework for Managing Drone BVLoS Operations*". In: IEEE Transactions on Network and Service Management (TNSM)[2].

2. Francesco Betti Sorbelli, Federico Corò, Sajal K Das, **Lorenzo Palazzetti**, Cristina M Pinotti, "*Drone-based Bug Detection in Orchards with Nets: A Novel Orienteering Approach*". In: ACM Transactions on Sensor Networks (TOSN)[3].

3. Francesco Betti Sorbelli, **Lorenzo Palazzetti**, Cristina M Pinotti, "*YOLO-based Detection of Halyomorpha halys in Orchards Using RGB Cameras and Drones*". In: Computers and Electronics in Agriculture (CEA).

4. Francesco Betti Sorbelli, Alfredo Navarra, **Lorenzo Palazzetti**, Cristina M. Pinotti, and Giuseppe Prencipe. "*Wireless IoT Sensors Data Collection Reward Maximization by Leveraging Multiple Energy-and Storage-Constrained UAVs*". Journal of Computer and System Sciences (JCSS), 2023.

5. Francesco Betti Sorbelli, Federico Corò, Sajal K Das, **Lorenzo Palazzetti**, Cristina M Pinotti. "*How the Wind Can Be Leveraged for Saving Energy in a Truck-Drone Delivery System*". IEEE Transactions on Intelligent Transportation Systems (T-ITS).

6. Francesco Betti Sorbelli, Federico Corò, Sajal K Das, **Lorenzo Palazzetti**, Cristina M Pinotti. "*On the Scheduling of Conflictual Deliveries in a last-mile delivery scenario with truck-carried drones*". In: Pervasive and Mobile Computing (PMC) (2022).

[2]This paper is not mentioned along the dissertation, and is currently under review.
[3]Currently under review with minor comments.

## Conference and Workshop Proceedings

1. Francesco Betti Sorbelli, **Lorenzo Palazzetti**, Cristina M Pinotti, "*A Drone-Based Automated Halyomorpha Halys Scouting: A Case Study on Orchard Monitoring*". In: IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor) (2023)

2. Francesco Betti Sorbelli, **Lorenzo Palazzetti**, Cristina M Pinotti, "*Preliminary Results for Halyomorpha Halys Monitoring Relying on a Custom Dataset*". In: IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor) (2023).

3. Dinca Alexandru, Popescu Dan, Maria Cristina Pinotti, Ichim Loretta, **Lorenzo Palazzetti**, Angelescu Nicoleta, "*Halyomorpha Halys Detection in Orchard from UAV Images Using Convolutional Neural Networks*". In: IWANN International Work Conference on Artificial Neural Networks (2023).

4. Lennart Almstedt, Davide Baltieri, Francesco Betti Sorbelli, Davide Cattozzi, Daniele Giannetti, Amin Kargar, Lara Maistrello, Alfredo Navarra, David Niederprüm, Brendan O'Flynn, **Lorenzo Palazzetti**, Niccolo Patelli, Luca Piccinini, Cristina M Pinotti, Lars Wolf, Dimitrios Zorbas, "*Technological Innovations in Agriculture for Scouting Halyomorpha halys in Orchards*". In: 5th International Workshop on Intelligent Systems for the Internet of Things (ISIOT) (2023).

5. Francesco Betti Sorbelli, Punyasha Chatterjee, Federico Corò, **Lorenzo Palazzetti**, Cristina M Pinotti, "*A Novel Multi-Layer Framework for BVLoS Drone Operation: A Preliminary Study*". In: IEEE INFOCOM DroneCom 2023 workshop (2023).[4]

6. Francesco Betti Sorbelli, Alfredo Navarra, **Lorenzo Palazzetti**, Cristina M Pinotti, Giuseppe Prencipe "*Optimal and Heuristic Algorithms for Data Collection by Using an Energy-and Storage-Constrained Drone*". 18th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, (ALGOSENSORS 2022) September 8-9, 2022.

7. Francesco Betti Sorbelli, Federico Corò, Sajal K. Das, Emanuele Di Bella, Lara Maistrello, **Lorenzo Palazzetti**, Cristina M. Pinotti "*A Drone-based Application for Scouting Halyomorpha halys Bugs in Orchards with Multifunctional Nets*". The 20th International Conference on Pervasive Computing and Communications (PerCom 2022) March 21-25, 2022.

8. Francesco Betti Sorbelli, Federico Corò, Sajal K. Das, **Lorenzo Palazzetti**, and Cristina M. Pinotti. "*Greedy Algorithms for Scheduling Package Delivery with Multiple Drones*". In: 23rd

[4]This paper is not mentioned along the dissertation.

International Conference on Distributed Computing and Networking (ICDCN), New Delhi, India, January 4-7, 2022.

9. Francesco Betti Sorbelli, Federico Corò, Sajal K Das, *Lorenzo Palazzetti*, Cristina M Pinotti "*Drone-based optimal and heuristic orienteering algorithms towards bug detection in orchards*". 18th Int. Conf. on Distr. Computing in Sensor Systems (DCOSS 2022) May 30 - June 1, 2022.

10. Francesco Betti Sorbelli, Federico Corò, Sajal K. Das, **Lorenzo Palazzetti**, and Cristina M. Pinotti. "*Cooperative Truck-Drone Scheduling Approach for Last-Mile Deliveries*". In: Italian Conference on Theoretical Computer Science (ICTCS short paper), September 13-15, 2021.

11. **Lorenzo Palazzetti**, Cristina M. Pinotti, and Giulio Rigoni. "*A run in the wind: Favorable winds make the difference in drone delivery*". 17th International Conference on Distributed Computing in Sensor Systems (DCOSS), July 14-16, 2021.

12. **Lorenzo Palazzetti**. "*Routing Drones Being Aware of Wind Conditions: a Case Study*". 17th International Conference on Distributed Computing in Sensor Systems (DCOSS), July 14-16, 2021.

# Bibliography


[1] Lennart Almstedt et al. Technological innovations in agriculture for scouting halyomorpha halys in orchards. In *2023 19th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 1–8, 2023.

[2] Georg Graetz and Guy Michaels. Robots at Work. *The Review of Economics and Statistics*, 100(5):753–768, 12 2018.

[3] Lorenzo Palazzetti. Routing drones being aware of wind conditions: a case study. In *2021 17th Intl. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 343–350. IEEE, 2021.

[4] Lorenzo Palazzetti, Cristina M Pinotti, and Giulio Rigoni. A run in the wind: favorable winds make the difference in drone delivery. In *17th Intl. Conf. on Distr. Comp. in Sensor Systems (DCOSS)*, pages 109–116. IEEE, 2021.

[5] Francesco Betti Sorbelli, Federico Corò, Lorenzo Palazzetti, Cristina M. Pinotti, and Giulio Rigoni. How the wind can be leveraged for saving energy in a truck-drone delivery system. *IEEE Transactions on Intelligent Transportation Systems*, 24(4):4038–4049, 2023.

[6] Francesco Betti Sorbelli, Federico Corò, Lorenzo Das, Sajal K Palazzetti, and Cristina M Pinotti. Cooperative truck-drone scheduling approach for last-mile deliveries. In *22nd Italian Conf. on Theoretical Computer Science (ICTCS)*, 2021.

[7] Francesco Betti Sorbelli, Federico Corò, Sajal Das, Lorenzo Palazzetti, and Cristina Pinotti. On the scheduling of conflictual deliveries in a last-mile delivery scenario with truck-carried drones. *Pervasive and Mobile Comp.*, page 101700, 2022.

[8] Pieter Vansteenwegen et al. The orienteering problem: A survey. *European Journal of Op. Research*, 209(1), 2011.

[9] Francesco Betti Sorbelli, Federico Corò, Sajal K Das, Emanuele Di Bella, Lara Maistrello, Lorenzo Palazzetti, and Cristina M Pinotti. A drone-based application for scouting halyomorpha halys bugs in orchards with multifunctional nets. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 127–129. IEEE, 2022.

[10] Francesco Betti Sorbelli, Federico Corò, Sajal K Das, Lorenzo Palazzetti, and Cristina M Pinotti. Drone-based optimal and heuristic orienteering algorithms towards bug detection in orchards. In *18th Intl. Conf. on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2022.

[11] Francesco Betti Sorbelli, Alfredo Navarra, Lorenzo Palazzetti, Cristina M Pinotti, and Giuseppe Prencipe. Optimal and heuristic algorithms for data collection by using an energy-and storage-constrained drone. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks*, pages 18–30, Cham, 2022. Springer International Publishing.

[12] Francesco Betti Sorbelli, Alfredo Navarra, Lorenzo Palazzetti, Cristina M. Pinotti, and Giuseppe Prencipe. Wireless iot sensors data collection reward maximization by leveraging multiple energy- and storage-constrained uavs. *Journal of Computer and System Sciences*, 139:103475, 2024.

[13] Kirtan Jha, Aalap Doshi, Poojan Patel, and Manan Shah. A comprehensive review on automation in agriculture using artificial intelligence. *Artificial Intelligence in Agriculture*, 2:1–12, 2019.

[14] Anne L Nielsen and George C Hamilton. Life history of the invasive species halyomorpha halys (hemiptera: Pentatomidae) in northeastern united states. *Annals of the Entomological Society of America*, 102(4):608–616, 2009.

[15] Lara Maistrello, Paride Dioli, Moreno Dutto, Stefania Volani, Sara Pasquali, and Gianni Gilioli. Tracking the spread of sneaking aliens by integrating crowdsourcing and spatial modeling: the italian invasion of *Halyomorpha halys*. *BioScience*, 68(12):979–989, 2018.

[16] Sibylle Stoeckli et al. Current distribution and voltinism of the brown marmorated stink bug, *Halyomorpha halys*, in Switzerland and its response to climate change using a high-resolution CLIMEX model. *International Journal of Biometeorology*, 64:2019–2032, 2020.

[17] HALY.ID. Project. `https://www.haly-id.eu`, 2022.

[18] Francesco Betti Sorbelli, Lorenzo Palazzetti, and Cristina M Pinotti. A drone-based automated halyomorpha halys scouting: A case study on orchard monitoring. In *2023 5th IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. IEEE, 2023.

[19] Francesco Betti Sorbelli, Lorenzo Palazzetti, and Cristina M Pinotti. Preliminary results for halyomorpha halys monitoring relying on a custom dataset. In *2023 5th IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. IEEE, 2023.

[20] Francesco Betti Sorbelli, Lorenzo Palazzetti, and Cristina M. Pinotti. Yolo-based detection of halyomorpha halys in orchards using rgb cameras and drones. *Computers and Electronics in Agriculture*, 213:108228, 2023.

[21] S Ahirwar, R Swarnkar, S Bhukya, and G Namwade. Application of drone in agriculture. *International Journal of Current Microbiology and Applied Sciences*, 8(1):2500–2505, 2019.

[22] Timothy S de Smet, Alex Nikulin, et al. Successful application of drone-based aeromagnetic surveys to locate legacy oil and gas wells in cattaraugus county, new york. *Journal of Applied Geophysics*, 186(C), 2021.

[23] Abegaz Mohammed Seid et al. Multi-agent drl for task offloading and resource allocation in multi-uav enabled iot edge network. *Trans. on Network and Service Management*, 18(4):4531–4547, 2021.

[24] Francesco Betti Sorbelli, Sajal K Das, Cristina M Pinotti, and Giulio Rigoni. A comprehensive investigation on range-free localization algorithms with mobile anchors at different altitudes. *Elsevier PMC*, 73:101383, 2021.

[25] Angelo Trotta, Fabio D Andreagiovanni, Marco Di Felice, et al. When uavs ride a bus: Towards energy-efficient city-scale video surveillance. In *INFOCOM 2018*, pages 1043–1051. IEEE, 2018.

[26] Francesco Betti Sorbelli, Cristina M Pinotti, and Giulio Rigoni. On the evaluation of a drone-based delivery system on a mixed euclidean-manhattan grid. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[27] David Schneider. The delivery drones are coming. *IEEE Spectrum*, 57(1):28–29, 2020.

[28] Brunilde Girardet, Laurent Lapasset, Daniel Delahaye, and Christophe Rabut. Wind-optimal path planning: Application to aircraft trajectories. In *13th Intl. Conf. on Control Automation Robotics & Vision*, pages 1403–1408. IEEE, 2014.

[29] Joshuah K Stolaroff, Constantine Samaras, Emma R O'Neill, et al. Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery. *Nature communications*, 9(1):1–13, 2018.

[30] Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

[31] Alena Otto, Niels Agatz, James Campbell, Bruce Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018.

[32] Nils Boysen, Stefan Fedtke, and Stefan Schwerdfeger. Last-mile delivery concepts: a survey from an operational research perspective. *OR Spectrum*, pages 1–58, 2020.

[33] Jacco M Hoekstra and Joost Ellerbroek. Bluesky atc simulator project: an open data and open source approach. In *7th Intl. Conf. on Research in Air Transportation*, volume 131, page 132. FAA/USA/Europe, 2016.

[34] Niels A.H. Agatz, Paul Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *ERIM Report Series Reference No. ERS-2015-011-LIS*, 2018.

[35] Júlia Cária de Freitas and Puca Huachi Vaz Penna. A variable neighborhood search for flying sidekick traveling salesman problem. *Intl. Trans. in Operational Research*, 27(1):267–290, 2020.

[36] Gloria Cerasela Crişan and Elena Nechita. On a cooperative truck-and-drone delivery system. *Procedia Computer Science*, 159:38–47, 2019.

[37] Nils Boysen, Dirk Briskorn, Stefan Fedtke, and Stefan Schwerdfeger. Drone delivery from trucks: Drone scheduling for given truck routes. *Networks*, 72(4):506–527, 2018.

[38] Ty Nguyen and Tsz-Chiu Au. Extending the range of delivery drones by exploratory learning of energy models. In *Autonomous Agents and MultiAgent Systems*, pages 1658–1660. IFAAMAS, 2017.

[39] Francesco Betti Sorbelli, Federico Corò, Sajal K Das, and Cristina M Pinotti. Energy-constrained delivery of goods with drones under varying wind conditions. *IEEE Trans. on Intelligent Transportation Systems*, 22(9):6048–6060, 2020.

[40] Arindam Khanda, Federico Corò, Francesco Betti Sorbelli, Cristina M Pinotti, and Sajal K Das. Efficient route selection for drone-based delivery under time-varying dynamics. In *2021 IEEE 18th Intl. Conf. on Mobile Ad Hoc and Smart Systems (MASS)*, pages 437–445. IEEE, 2021.

[41] Amila Thibbotuwawa, Grzegorz Bocewicz, Peter Nielsen, and Banaszak Zbigniew. Planning deliveries with uav routing under weather forecast and energy consumption constraints. *IFAC-PapersOnLine*, 52(13):820–825, 2019.

[42] Amila Thibbotuwawa, Grzegorz Bocewicz, Grzegorz Radzki, Peter Nielsen, and Zbigniew Banaszak. Uav mission planning resistant to weather uncertainty. *Sensors*, 20(2):515, 2020.

[43] Grzegorz Radzki, Peter Nielsen, Grzegorz Bocewicz, and Zbigniew Banaszak. A proactive approach to resistant uav mission planning. In *Conf. on Automation*, pages 112–124. Springer, 2020.

[44] Jermaine Janszen, Babar Shahzaad, et al. Constraint-aware trajectory for drone delivery services. In *ICSOC*. Springer, 2021.

[45] Thomas Kirschstein. Energy demand of parcel delivery services with a mixed fleet of electric vehicles. *Cleaner Engineering and Technology*, 5:100322, 2021.

[46] John M. Wallace and Peter V. Hobbs. *Atmospheric science: An introductory survey*. Amsterdam: Elsevier Academic Press., 2006.

[47] Wayne Johnson. *Helicopter theory*. Courier Corporation, 2012.

[48] Kenzo Nonami. Drone technology, cutting-edge drone business, and future prospects. *Journal of Robotics and Mechatronics*, 28(3):262–272, 2016.

[49] Ross Arnold et al. Experimentation for optimization of heterogeneous drone swarm configurations: terrain and distribution. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, page 1174625. International Society for Optics and Photonics, 2021.

[50] Jaihyun Lee. Optimization of a modular drone delivery system. In *2017 Annual IEEE Intl. Systems Conf., SysCon 2017, Montreal, QC, Canada, April 24-27, 2017*, pages 1–8. IEEE, 2017.

[51] Iman Dayarian, Martin Savelsbergh, and John-Paul Clarke. Same-day delivery with drone resupply. *Transportation Science*, 54(1):229–249, 2020.

[52] Neil Mathew, Stephen L Smith, and Steven L Waslander. Planning paths for package delivery in heterogeneous multirobot teams. *IEEE Transactions on Automation Science and Engineering*, 12(4):1298–1308, 2015.

[53] Sungwoo Kim and Ilkyeong Moon. Traveling salesman problem with a drone station. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):42–52, 2018.

[54] Rami Daknama and Elisabeth Kraus. Vehicle routing with drones. *CoRR*, abs/1705.06431, 2017.

[55] Roberto Roberti and Mario Ruthmair. Exact methods for the traveling salesman problem with drone. *Transportation Science*, 55(2):315–335, 2021.

[56] Desheng Wang et al. Routing and scheduling for hybrid truck-drone collaborative parcel delivery with independent and truck-carried drones. *Internet of Things Journal*, 6(6):10483–10495, 2019.

[57] Yong Sik Chang and Hyun Jung Lee. Optimal delivery routing with wider drone-delivery areas along a shorter truck-route. *Expert Systems with Applications*, 104:307–317, 2018.

[58] Chase C Murray and Ritwik Raj. The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. *Transportation Research Part C: Emerging Technologies*, 110:368–398, 2020.

[59] Silvano Martello, David Pisinger, and Paolo Toth. New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research*, 123(2):325–332, 2000.

[60] Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[61] Timothy M Chan. Approximation schemes for 0-1 knapsack. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, page 1. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[62] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000.

[63] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical programming*, 14(1):265–294, 1978.

[64] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[65] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[66] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[67] John M. Harris, Jeffry L. Hirst, and Michael J. Mossinghoff. *Combinatorics and Graph Theory, Second Edition*. Undergraduate Texts in Mathematics. Springer, 2008.

[68] Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49(5):640–671, September 2002.

[69] David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974.

[70] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[71] Muhammad Qasim and Csiszár Csaba. Major barriers in adoption of electric trucks in logistics system. *Promet-Traffic&Transportation*, 33(6):833–846, 2021.

[72] Francesco Betti Sorbelli, Federico Corò, Sajal K. Das, and Cristina M. Pinotti. Energy-constrained delivery of goods with drones under varying wind conditions. *IEEE Trans. Intell. Transp. Syst.*, 22(9):6048–6060, 2021.

[73] Catriona Tullo and James Hurford. Modelling zipfian distributions in language. In *Proceedings of language evolution and computation workshop/course at ESSLLI*, pages 62–75, 2003.

[74] GK Sujayanand, S Sheelamary, and G Prabhu. Recent innovations and approaches for insect pest management in agriculture. *Biotica Research Today*, 3(2):100–102, 2021.

[75] Sabbatini Peverieri Giuseppino, Bortolotti Pier Paolo, Nannini Roberta, Marianelli Leonardo, and Roversi Pio Federico. Efficacy of long lasting insecticide nets in killing halyomorpha halys in pear orchards. *Outlooks on Pest Management*, 29(2):70–74, 2018.

[76] Ina Schlathölter, Anna Dalbosco, Michael Meissle, Andrea Knauf, Alex Dallemulle, Beat Keller, Jörg Romeis, Giovanni AL Broggini, and Andrea Patocchi. Low outcrossing from an apple field trial protected with nets. *Agronomy*, 11(9):1754, 2021.

[77] James Christopher Bergh et al. Border habitat effects on captures of halyomorpha halys (hemiptera: Pentatomidae) in pheromone traps and fruit injury at harvest in apple and peach orchards in the mid-atlantic, usa. *Insects*, 12(5):419, 2021.

[78] Paula M Shrewsbury and Michael J Raupp. Evaluation of components of vegetational texture for predicting azalea lace bug, stephanitis pyrioides (heteroptera: Tingidae), abundance in managed landscapes. *Environmental Entomology*, 29(5):919–926, 2000.

[79] Richard Weinzierl and Tess Henn. Botanical insecticides and insecticidal soaps. In *Handbook of integrated pest management for turf and ornamentals*, pages 541–555. CRC Press, 2020.

[80] Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[81] Avrim Blum, Shuchi Chawla, David R Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *SIAM Journal on Comp.*, 37(2):653–670, 2007.

[82] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Op. Res.*, 255(2):315–332, 2016.

[83] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 46–55, 2003.

[84] Ke Chen and Sariel Har-Peled. The orienteering problem in the plane revisited. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, page 247–254, 2006.

[85] Thomas C Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Routing algorithms for robot assisted precision irrigation. In *Intl. Conf. on Robotics and Automation (ICRA)*, pages 2221–2228. IEEE, 2018.

[86] Thomas C Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Multi-robot routing algorithms for robots operating in vineyards. In *2018 IEEE 14th Intl. Conf. on Automation Science and Engineering (CASE)*, pages 14–21. IEEE, 2018.

[87] Francesco Betti Sorbelli, Stefano Carpin, Federico Corò, Alfredo Navarra, and Cristina M Pinotti. Optimal routing schedules for robots operating in aisle-structures. In *2020 IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4927–4933. IEEE, 2020.

[88] Francesco Betti Sorbelli, Federico Corò, Sajal K Das, Alfredo Navarra, and Cristina M Pinotti. Speeding-up routing schedules on aisle-graphs. In *16th Intl. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 69–76. IEEE, 2020.

[89] Francesco Betti Sorbelli, Stefano Carpin, Federico Corò, Sajal K Das, Alfredo Navarra, and Cristina M Pinotti. Speeding up routing schedules on aisle graphs with single access. *IEEE Transactions on Robotics*, 2021.

[90] Thomas C Thayer and Stefano Carpin. An adaptive method for the stochastic orienteering problem. *IEEE Robotics and Automation Letters*, 6(2):4185–4192, 2021.

[91] Xinyue Kan, Thomas C Thayer, Stefano Carpin, and Konstantinos Karydis. Task planning on stochastic aisle graphs for precision agriculture. *IEEE Robotics and Automation Letters*, 6(2):3287–3294, 2021.

[92] Stefano Carpin and Thomas C Thayer. Solving stochastic orienteering problems with chance constraints using monte carlo tree search. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 1170–1177. IEEE, 2022.

[93] Mike Roberts, Debadeepta Dey, Anh Truong, Sudipta Sinha, Shital Shah, Ashish Kapoor, Pat Hanrahan, and Neel Joshi. Submodular trajectory optimization for aerial 3d scanning. In *Proceedings of the IEEE Intl. Conf. on Computer Vision*, pages 5324–5333, 2017.

[94] Erin E. Grabarczyk, Ted E. Cottrell, and P. Glynn Tillman. Spatiotemporal distribution of halyomorpha halys (stål) (hemiptera: Pentatomidae) across a fruit and tree nut agricultural ecosystem. *Environmental Entomology*, 51(4):824 – 835, 2022. Cited by: 7; All Open Access, Bronze Open Access.

[95] Noel G. Hahn, Cesar Rodriguez-Saona, and George C. Hamilton. Characterizing the spatial distribution of brown marmorated stink bug, halyomorpha halys stål (hemiptera: Pentatomidae), populations in peach orchards. *PLOS ONE*, 12:1–20, 03 2017.

[96] Joanna J Fisher, Jhalendra P Rijal, and Frank G Zalom. Temperature and Humidity Interact to Influence Brown Marmorated Stink Bug (Hemiptera: Pentatomidae), Survival. *Environmental Entomology*, 50(2):390–398, 12 2020.

[97] Andreas Krause and Carlos Guestrin. *A note on the budgeted maximization of submodular functions*. Carnegie Mellon University. Center for Automated Learning and Discovery, 2005.

[98] Mengyu Chen, Weifa Liang, and Yuchen Li. Data collection maximization for uav-enabled wireless sensor networks. In *29th Intl. Conf. on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2020.

[99] Mengyu Chen, Weifa Liang, and Jing Li. Energy-efficient data collection maximization for uav-assisted wireless sensor networks. In *Wireless Communications and Networking Conf. (WCNC)*, pages 1–7. IEEE, 2021.

[100] Mengyu Chen, Weifa Liang, and Sajal K Das. Data collection utility maximization in wireless sensor networks via efficient determination of uav hovering locations. In *PerCom*, pages 1–10. IEEE, 2021.

[101] Junqi Zhang, Zheng Li, Wenzheng Xu, Jian Peng, Weifa Liang, Zichuan Xu, Xiaojiang Ren, and Xiaohua Jia. Minimizing the number of deployed uavs for delay-bounded data collection of iot devices. In *INFOCOM*, pages 1–10. IEEE, 2021.

[102] Cheng Zhan, Yong Zeng, and Rui Zhang. Energy-efficient data collection in uav enabled wireless sensor network. *IEEE Wireless Communications Letters*, 7(3):328–331, 2017.

[103] Rezoan Ahmed Nazib and Sangman Moh. Energy-efficient and fast data collection in uav-aided wireless sensor networks for hilly terrains. *IEEE Access*, 9:23168–23190, 2021.

[104] Juan Liu, Peng Tong, Xijun Wang, Bo Bai, and Huaiyu Dai. Uav-aided data collection for information freshness in wireless sensor networks. *IEEE Trans. on Wireless Communications*, 20(4):2368–2382, 2020.

[105] Xiong Li, Jiawei Tan, et al. A novel uav-enabled data collection scheme for intelligent transportation system through uav speed control. *IEEE Trans. on Intelligent Transportation Systems*, 22(4), 2020.

[106] Mahdi Ben Ghorbel, David Rodriguez-Duarte, Hakim Ghazzai, Md Jahangir Hossain, and Hamid Menouar. Energy efficient data collection for wireless sensors using drones. In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2018.

[107] Rone Ilídio da Silva and Mario A Nascimento. On best drone tour plans for data collection in wireless sensor network. In *Proceedings of the 31st annual ACM symposium on applied computing*, pages 703–708, 2016.

[108] Cong Pu, Andrew Wall, Imtiaz Ahmed, and Kim-Kwang Raymond Choo. Secureiod: A secure data collection and storage mechanism for internet of drones. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, pages 83–92. IEEE, 2022.

[109] Angelo Trotta, Marco Di Felice, Luciano Bononi, Enrico Natalizio, Luca Perilli, Eleonora Franchi Scarselli, Tullio Salmon Cinotti, and Roberto Canegallo. Bee-drones: Energy-efficient data collection on wake-up radio-based wireless sensor networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 547–553. IEEE, 2019.

[110] Dimitrios Zorbas and Brendan O'Flynn. Collision-free sensor data collection using lorawan and drones. In *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–5. IEEE, 2018.

[111] Rone Ilídio da Silva, Josiane Da Costa Vieira Rezende, and Marcone Jamilson Freitas Souza. Collecting large volume data from wireless sensor network by drone. *Ad Hoc Networks*, 138:103017, 2023.

[112] I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996.

[113] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

[114] Silvano Martello and Paolo Toth. Solution of the zero-one multiple knapsack problem. *European Journal of Operational Research*, 4(4):276–283, 1980.

[115] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[116] Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 313–324. Springer, 2012.

[117] Silvano Martello and Paolo Toth. Heuristic algorithms for the multiple knapsack problem. *Computing*, 27(2):93–112, 1981.

[118] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[119] Jaya Sreevalsan-Nair. K-means clustering. In *Encyclopedia of Mathematical Geosciences*, pages 1–3. Springer, 2021.

[120] Feiping Nie, Ziheng Li, Rong Wang, and Xuelong Li. An effective and efficient algorithm for k-means clustering with new formulation. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[121] Janis Jansons and Toms Dorins. Analyzing ieee 802.11 n standard: outdoor performanace. In *2012 Second International Conference on Digital Information Processing and Communications (ICDIPC)*, pages 26–30. IEEE, 2012.

[122] Aakash Khochare, Yogesh Simmhan, Francesco Betti Sorbelli, and Sajal K. Das. Heuristic algorithms for co-scheduling of edge analytics and routes for uav fleet missions. In *IEEE INFOCOM 2021 - IEEE Conf. on Computer Communications*, pages 1–10, 2021.

[123] A Sava, L Ichim, and D Popescu. Detection of halyomorpha halys using neural networks. In *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 437–442. IEEE, 2022.

[124] Abderahman Rejeb, Alireza Abdollahi, Karim Rejeb, and Horst Treiblmaier. Drones in agriculture: A review and bibliometric analysis. *Computers and electronics in agriculture*, 198:107017, 2022.

[125] Yiqing Guo, Xiuping Jia, David Paull, Junpeng Zhang, Adnan Farooq, Xiaolin Chen, and Md. Nazrul Islam. A drone-based sensing system to support satellite image analysis for rice farm mapping. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 9376–9379, 2019.

[126] Deepak Murugan, Akanksha Garg, Tasneem Ahmed, and Dharmendra Singh. Fusion of drone and satellite data for precision agriculture monitoring. In *2016 11th International Conference on Industrial and Information Systems (ICIIS)*, pages 910–914, 2016.

[127] L. Moreno, V. Ramos, M. Pohl, and F. Huguet. Comparative study of multispectral satellite images and rgb images taken from drones for vegetation cover estimation. In *2018 IEEE 38th Central America and Panama Convention (CONCAPAN XXXVIII)*, pages 1–8, 2018.

[128] Suriyon Tansuriyavong, Hideto Koja, Motoki Kyan, and Takashi Anezaki. The development of wildlife tracking system using mobile phone communication network and drone. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 3, pages 351–354, 2018.

[129] Ankush Mitra, Basudeb Bera, and Ashok Kumar Das. Design and testbed experiments of public blockchain-based security framework for iot-enabled drone-assisted wildlife monitoring. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021.

[130] Andrés Montes de Oca and Gerardo Flores. The agriq: A low-cost unmanned aerial system for precision agriculture. *Expert Systems with Applications*, 182:115163, 2021.

[131] Chengjun Xie, Jie Zhang, Rui Li, Jinyan Li, Peilin Hong, Junfeng Xia, and Peng Chen. Automatic classification for field crop insects via multiple-task sparse representation and multiple-kernel learning. *Computers and Electronics in Agriculture*, 119:123–132, 2015.

[132] Karlos Espinoza, Diego L Valera, José A Torres, Alejandro López, and Francisco D Molina-Aiz. Combination of image processing and artificial neural networks as a novel approach for the identification of *Bemisia tabaci* and *Frankliniella occidentalis* on sticky traps in greenhouse agriculture. *Computers and Electronics in Agriculture*, 127:495–505, 2016.

[133] Miroslav Valan, Karoly Makonyi, Atsuto Maki, Dominik Vondráček, and Fredrik Ronquist. Automated taxonomic identification of insects with expert-level accuracy using effective feature transfer from convolutional networks. *Systematic Biology*, 68(6):876–895, 2019.

[134] Chenglu Wen and Daniel Guyer. Image-based orchard insect automated identification and classification method. *Computers and electronics in agriculture*, 89:110–115, 2012.

[135] Ching-Ju Chen, Ya-Yu Huang, Yuan-Shuo Li, Chuan-Yu Chang, and Yueh-Min Huang. An aiot based smart agricultural system for pests detection. *IEEE Access*, 8:180750–180761, 2020.

[136] Yue He, Zhiyan Zhou, Luhong Tian, Youfu Liu, and Xiwen Luo. Brown rice planthopper (*Nilaparvata lugens* (Stål)) detection based on deep learning. *Precision Agriculture*, 21(6):1385–1402, 2020.

[137] Wenyong Li, Dujin Wang, Ming Li, Yulin Gao, Jianwei Wu, and Xinting Yang. Field detection of tiny pests from sticky trap images using deep learning in agricultural greenhouse. *Computers and Electronics in Agriculture*, 183:106048, 2021.

[138] Yong-Lak Park, Jum Rae Cho, Gwan-Seok Lee, and Bo Yoon Seo. Detection of *Monema flavescens* (Lepidoptera: Limacodidae) cocoons using small unmanned aircraft system. *Journal of Economic Entomology*, 114(5):1927–1933, 2021.

[139] Veronica Ferrari, Rosalba Calvini, Bas Boom, Camilla Menozzi, Aravind Krishnaswamy Rangarajan, Lara Maistrello, Peter Offermans, and Alessandro Ulrici. Evaluation of the potential of near infrared hyperspectral imaging for monitoring the invasive brown marmorated stink bug. *Chemometrics and Intelligent Laboratory Systems*, page 104751, 2023.

[140] Raluca Trufelea, Mihai Dimoiu, Loretta Ichim, and Dan Popescu. Detection of harmful insects for orchard using convolutional neural networks. *UPB Sci. Bull. Ser. C*, 83(4):85–96, 2021.

[141] DJI. Maryland Biodiversity Dataset. https://www.marylandbiodiversity.com/, 2022.

[142] L Ichim, R Ciciu, and D Popescu. Using drones and deep neural networks to detect halyomorpha halys in ecological orchards. In *IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium*, pages 437–440. IEEE, 2022.

[143] G Jocher et al. ultralytics/yolov5: v6. 1-tensorrt, tensorflow edge tpu and openvino export and inference, 2022.

[144] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[145] Niranjan D. Narvekar and Lina J. Karam. A no-reference image blur metric based on the cumulative probability of blur detection (cpbd). *IEEE Transactions on Image Processing*, 20(9):2678–2683, 2011.

[146] Colin Ware. Chapter three - lightness, brightness, contrast, and constancy. In Colin Ware, editor, *Information Visualization (Fourth Edition)*, Interactive Technologies, pages 69–94. Morgan Kaufmann, fourth edition edition, 2021.

[147] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[148] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[149] Tsung-Yi Lin et al. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755, Zurich, Switzerland, 2014. Springer.

[150] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *19th Intl. Conference on Computational Statistics, COMPSTAT*, pages 177–186, Paris, 2010. Physica-Verlag.

[151] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[152] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: Solution for Edge Computing Applications*, pages 87–104, 2021.

[153] Piotr Skalski. Make Sense. https://github.com/SkalskiP/make-sense/, 2019.

[154] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[155] Fuzhen Zhuang et al. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.

[156] Buslaev A., Parinov A., Khvedchenya E., Iglovikov V. I., and A. A. Kalinin. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*, 2018.

[157] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021.

[158] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[159] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection, 2022.

[160] Lara Maistrello, Giacomo Vaccari, Stefano Caruso, Elena Costi, Sara Bortolini, Laura Macavei, Giorgia Foca, Alessandro Ulrici, Pier Paolo Bortolotti, Roberta Nannini, et al. Monitoring of the invasive *Halyomorpha halys*, a new key pest of fruit orchards in northern Italy. *Journal of Pest Science*, 90:1231–1244, 2017.