SAPIENZA
UNIVERSITÀ DI ROMA

Towards Autonomous Networks of Drones

Phd School in Computer Science

Dottorato di Ricerca in Computer Science – XXXIV Ciclo

Candidate

Andrea Coletta

ID number 1598709

Thesis Advisor

Prof. Gaia Maselli

January 2022

Thesis defended on 20 May 2022
in front of a Board of Examiners composed by:

Prof. Gabriella Pasi

Prof. Mauro Conti

Prof. Salvatore Gaglio

Towards Autonomous Networks of Drones
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: March 23, 2023

Author's email: coletta@di.uniroma1.it

Abstract

Unmanned Aerial Vehicles (UAVs) have enormous potential in the public and civilian domain as an emerging technology that allows to carry out missions where human intervention would be impossible, unsafe, or just inefficient. While single remotely controlled drones are used in a wide range of scenarios, an extraordinary improvement could be reached by employing autonomous networks of drones.

Current work in this direction highlights significant issues caused by the dynamic topology and the limited energy of the drones, especially in safety-critical missions. In this thesis, we investigate solutions and methodologies for the coordination and communication of UAVs to enable autonomous networks of drones in real applications. We focus on drones' physical constraints and we study algorithms' applicability in real field.

We first address the problem of trajectories planning for safety-critical operations and we propose several solutions including approximation and genetic-based algorithms. Then we exploit the drones' communication capabilities to design efficient algorithms in a broader range of scenarios. We consider missions with uncertain target positions, in which drones coordinate by sharing their local observations, and missions in which drones offload the collected data to the depot through a multi-hop connection.

To improve drone communication capabilities, we propose an enhanced Routing protocol with Deep Q-Learning. The protocol is designed to work on critical scenarios requiring urgent communication that can tolerate only a bounded delay.

Finally, we investigate three novel applications for UAV networks to demonstrate how drones can populate our cities in the next years. We design DRUBER, a distributed parcel delivery system aided by a blockchain framework; we propose DANGER, an emergency network of drones to provide WiFi connection to any smartphone in case of disasters; and we discuss an application of drones for Food Safety and Security.

Contents

Chapter 1

Introduction

This thesis investigates the use of networks of Unmanned Aerial Vehicles (UAVs) in safety critical scenarios, proposing solutions and methodologies to fill the current gap in state-of-art work. In details we study applications, communication protocols, and coordination and control algorithms for aerial drones.

1.1 Motivation

UAVs or drones are air-crafts without human pilot aboard. Originally designed for military missions in which human intervention was too dangerous, drones are now commonly used in a wide range of public and civilian applications.

Drones are being used for rescue operations [1], remote health care [2], agriculture-crop survey [3], wildlife search [4], or parcel delivery [5]. Drone technology is growing rapidly with an estimated market value of around 46 billion dollars in the United States by 2026 [6].
The fundamental benefit of UAVs is the combination of powerful sensor equipment (*e.g.,* cameras, radars, and GPS), wireless networking, and mobility on the same device, making them perfect for applications that need little to no human control. While remotely controlled drones are widely used, squads of multiple autonomous and cooperative drones are still rare due to their high complexity of control, coordination and deployment [7]. This thesis provides new solutions to enable autonomous network of drones, unleashing their potential in terms of scalability, cost, failure probability, and speed.

Prior work for multi-UAVs networks highlights significant issues caused by their dynamic topology and limited energy, and provide preliminary solutions for safe and reliable communication, routing algorithms, and drones' applications [8–13]. However these solutions do not consider several issues related to coordination and control, which must be addressed before UAVs can be deployed in real applications.

In this thesis we study squads of drones and we propose new solutions and methodologies for UAVs coordination and communication. We mostly focus on safety critical scenarios, as fleets of aerial drones are a powerful tool to support the rescue operations. The Horizon Europe Work Program recently envisioned the use of drones to enable situation awareness in critical areas and provide a more appropriate and timely intervention [14].

1.2 Contribution

This thesis improves the current state-of-art for multi-UAVs by addressing the following problems:

Trajectory Planning. We first address the task-assignment and trajectory planning in critical settings (e.g., search-and-rescue of survivors, medicines distribution, water delivery to trapped or disabled humans or animals). We consider the timeliness of intervention a strict requirement, and we develop two automated trajectory planning algorithms to provide early inspection of target locations. We consider a variant of the traveling salesman problem [15] for multi-UAV, with known target locations. Conversely from existing literature [16, 17], we consider drones' physical constraints (e.g., battery lifetime) and we study the algorithms' applicability in a real test-bed. In Section 2.1 we formally analyze the problem, we propose a Mixed-Integer Linear Programming formulation and two related approximation algorithms, namely AC-GaP and TC-GaP [18, 19]. We demonstrate a constant factor approximation of 1/2 from the optimal solution, and superior performance with respect to the existing work. In Section 2.2 we propose Gen-Path [20], a genetic-based approach able to adapt to a wider range of scenarios.

Connected Trajectory Planning. Safety-critical missions are often executed in highly dynamic environments requiring, for example, continuous communication with the control center to adapt the mission upon local findings. Therefore, to address a wider range of scenarios, we exploit the communication capabilities of drones in the design of trajectory planning algorithms. In section 3.1 we propose GCD [21], a Greedy Connected Deployment algorithm to address a monitoring application in which drones have to communicate the collected data while inspecting the targets. Conversely from existing literature, we do not rely on any communication infrastructures (e.g., cellular networking) which may be disrupted in harsh environments [22], but we propose a novel solution to create connected formations and ensure multi-hop low-latency communication. We improve the state-of-art solutions by visiting about $15 - 20\%$ more target locations.
In section 3.2 we relax the assumption of perfect knowledge of ongoing events and their location. We consider a safety critical application in which the events' time and position can only be estimated with some *uncertainty* (e.g., survivors after an earthquake). We propose SIDE [23], a virtual-force based approach to let drones autonomously inspect an area of interest under *uncertainty*. In particular, drones leverage communication capabilities to share their local observations of the environment and coordinate during the exploration. We show that our proposal discovers new events $30 - 40\%$ faster than existing algorithms.

Communication. Drones are invaluable tools to build fast and reliable networks. However, due to their unconstrained mobility and unique characteristics, existing communication protocols designed for Mobile Ad-hoc NETworks (MANETs) fail short to provide stable communication. In Section 4.1 we propose MAD [24], an enhanced Routing protocol with Deep Q-Learning, that considers the unique characteristics of UAV networks like: limited energy, high mobility, dynamic topologies, and the varying requirements of safety critical application. The protocol is designed to work on critical applications requiring urgent communication that can tolerate only a

bounded delay. It enables adaptive selection of the most suitable relay nodes for packet delivery, resorting to movement-assisted delivery upon need, supported by a reinforcement learning approach. It doubles the packets delivered by classical geographical approaches, while it deliveries around $20 - 30\%$ more packets than existing movement-assisted protocols.

Applications. Finally, we investigate three novel applications for UAVs networks. In Section 5.1 we propose and implement DANGER [25], an emergency network of drones to provide WiFi connection to any smartphone in case of disasters. In Section 5.2 we design DRUBER [26] a distributed parcel delivery system aided by a blockchain framework. DRUBER proposes a fully distributed service based on a fleet of coordinated drones, belonging to multiple owners. The service provides a parcel delivery, with intermediate pit stops for battery replacement or drone-to-drone parcel handovers. Finally, we discuss the application of drones for Food Safety and Security. Drones can be used to improve farms performance by seeding, fertilizing, diagnosing diseases, monitoring crop and soil health [27]. In Section 5.3 we propose a crowd-sensing framework [28, 29], to detect plant diseases and analyze farmers' performance, and we discuss how the use of drones can improve the framework performance.

1.3 Background

In this section we present a brief introduction to the used methodologies.
We first introduce *Integer Linear Programming (ILP)* and *Mixed Integer Linear Programming (MILP)* mathematical formulations, which are used in Section 2.1, 3.1, 3.2 and Section 5.3. We then discuss modern machine learning algorithms, in particular *Reinforcement Learning (RL)* and *Deep Reinforcement Learning (DRL)*, used in Section 4.1. Finally, we briefly discuss the blockchain technology used in Section 5.2.

1.3.1 Optimization Problems

In a wide range of real-life applications we face hard problems which have no evident best solution. An optimization problem try to formally define the set of all feasible solutions and find the best solution for such problems.

Linear programming (LP). We first introduce the concept of *Linear programming (LP)* which is commonly used in commercial activities to maximize the income and minimize costs. For example, we can consider a LP model for a winery that produces two types of wine, *red* and *white*. We can assume that a liter of *red* wine requires 3 hours to be produced, while a liter of *white* requires only 2 hours. Each liter is sold at 22$ and 14.5$, respectively. Considering 40 working hours per week, the winery wants to maximize its profit. Therefore, we can define a LP optimization problem as follows:

$$\begin{array}{ll} \text{maximize} & 22 \cdot X + 14.5 \cdot Y \\ \text{subject to} & 3 \cdot X + 2 \cdot Y \qquad \leq 40 \\ & X, Y \in \mathcal{R} \end{array}$$

We define the amount of *red* and *white* wine to produce using the *continuous decision variables* X and Y, respectively. We *constraint* the maximum working hours per week with the linear constraint $3 \cdot X + 2 \cdot Y \leq 40$, which defines the set of feasible solutions. And finally we formulate an *objective function* to maximize the profit: we consider the total liters of *red* and *white* produced, multiplied by their per liter profit (i.e., $22 \cdot X + 14.5 \cdot Y$).

The optimization problem is solved by means of specialized algorithms, like the Simplex, and the values of variables X and Y define the winery strategy for the production.

Integer Linear Programming (ILP). An *ILP* formulation is a special case of LP in which all the variables are integers. In contrast to LP, these problems are intrinsically hard to solve and NP-hard in most cases. A classic example of an ILP optimization problem is the Traveling Salesman Problem: *Given a set of cities and the distances between each city, what is the shortest possible route that visits each city exactly once and returns to the origin location?*
We can define an ILP formulation for the Traveling Salesman Problem as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, && \forall i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} = 1, && \forall j = 1, \ldots, n \\
& \sum_{i,j \in S} x_{ij} \leq |S| - 1, && \forall S \subset \{1, \ldots, n\} : 2 \leq |S| \leq n - 1 \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

where the set of integer decision variables x_{ij} defines the salesman tour: $x_{ij} = 1$ if the salesman travels from city i to city j, and $x_{ij} = 0$ otherwise. The variable c_{ij} defines the cost to travel from city i to j. The first two constraints ensure that each city i is visited exactly once, while the last constraint imposes a single unique tour. Also this optimization problem can be solved by means of specialized algorithms, like Branch-and-Bound, and the values of x_{ij} variables define the salesman tour to visit all the cities.

Mixed Integer Linear Programming (ILP). MILP formulations extend *ILP* formulations by considering both integer and continuous variables. Generally, MILP formulations are even harder to solve.

Solution Methods. Several algorithms have been designed to solve such mathematical formulations, including Simplex, Branch-and-Bound, Branch-and-cut algorithms. In this thesis we use Gurobi [30], a fast and powerful solver which provides a collection of algorithms to solve LP, ILP and MILP formulations.

1.3.2 Reinforcement Learning

Reinforcement Learning (RL) [31] addresses the fundamental issue of how an intelligent agent can learn a good sequence of decisions. Based on the observation of an environment, the agent can take an action that affects the environment and itself, and depending on the outcome of the action the agent receives a *reward* that can

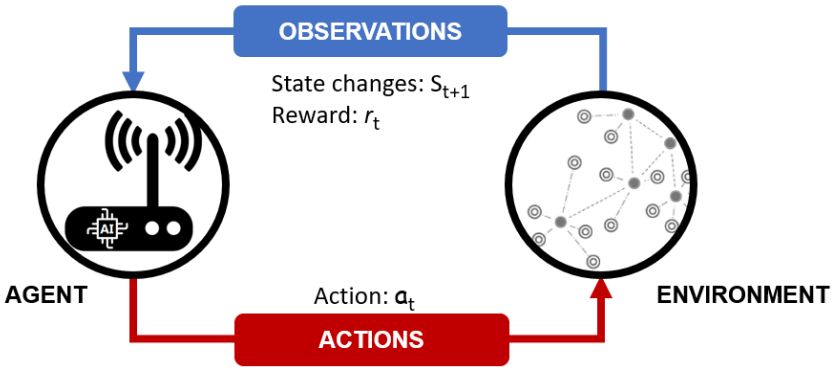


Figure 1.1. Agent-environment interaction in Reinforcement Learning.

be positive or negative. By exploring the different actions and their outcomes, the agent can learn the best sequence of decisions to maximize the positive rewards over time.

Figure 1.1 represents the agent's interaction with the environment. The agent interacts with its environment in discrete time steps: it executes an action a_t at time step t that alters the environment state, from s_t to s_{t+1}. The agent observes the new environment and obtains a reward r_t for the undertaken action. The goal of the RL agent is to learn the optimal policy and maximize the obtained rewards in the future.

Formally, a RL problem can be modeled as a Markov Decision Process (MDP) [32].

Definition 1.3.1 (Markov Decision Process)**.** *A Markov Decision Process (MDP) is a 4-tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$*, where:*

- $\mathcal{S}$ *is the state space*
- $\mathcal{A}$ *is the action space*
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ *is the transition function, where with* $\Delta(\mathcal{S})$ *we indicate the space of probability distributions over* $\mathcal{S}$*.*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function*

The RL agent learns how to act in order to maximize the amount of rewards earned, trying to learn an optimal policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which maximize the expected return defined as:

$$\mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}(s_t, a_t)\right]$$

where T is the horizon of the problem, which can be infinite or a positive integer, and $\gamma \in [0, 1]$ is the discount factor to prioritize immediate vs. future rewards.

The policy map π gives the probability of taking action $a_t \in A$ in the state $s_t \in S$.

Q-Learning. RL algorithms can be divided into model-based and model-free or, in parallel, in value-based, policy-search or actor-critic. *Model-free* algorithms

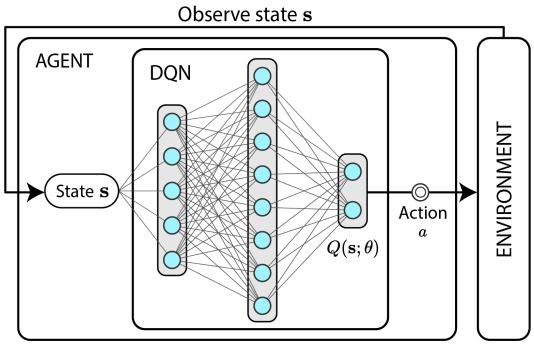


Figure 1.2. Deep Q-Network (DQN)

learn from the interactions with the environment, and the most common examples are Q-learning [33] and SARSA [34].

Q-learning is one of the most important breakthroughs in reinforcement learning, it learns an optimal policy π by updating an action-value function in a convergent iterative process that exploits the Bellman equation. According the Bellman Equation we update a state–action pair value $Q(s_t, a_t)$ by looking at the immediate reward $R(s_t, a_t)$ plus the maximum cumulative future discounted reward for the state $\mathbf{s_{t+1}}$ the agents ends up to.

The update rule is formally described as follows:

$$\underbrace{Q(s_t, a_t)}_{\text{New Q-Value}} = Q(s_t, a_t) + \underbrace{\alpha}_{\text{learning rate}} [\underbrace{R(s_t, a_t)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount factor}} \overbrace{\max_a Q(s_{t+1}, a)}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s_t, a_t)]$$

Where α is the learning rate, which controls how quickly we learn, and γ is the discount factor. After an exploration phase, the state–action pair values $Q(s, a)$ define the optimal policy π, and they are used to select the best action in a given state s.

Deep Reinforcement Learning (DRL). DRL is the combination of Reinforcement Learning with Deep Learning methods. In practice using Q-learning to estimate an optimal policy is complicated due to the dynamic programming nature of the process, which requires storing into memory the quality values of $|\mathcal{S}| \cdot |\mathcal{A}|$ pairs. For example, when RL was exploited to play Atari games in [35] the state space had size $\approx 10^{67970}$, which is the number of rows of an hypothetical table of quality values. Even if it is just a sparse table, the dimensionality curse would make the convergence of the iterative process impossible.

DRL leverages the potential of Deep Learning to solve the decision making problems of Reinforcement Learning more efficiently. Neural Networks can create great approximations of functions and extract information from high-dimensional rough data. They can approximate both the model of the environment; the policy function; or the state-action values.

Deep Q-Network (DQN). The most famous DRL algorithm is DQN, a value-based method that tries to approximate the Q-function using a Deep Neural Network. The DQN produces $Q(\mathbf{s};\theta) \approx Q^*(\mathbf{s})$ with θ being the weights of the network. The DQN takes the state s of the environment as input, and it outputs a vector that represents the Q values for each possible action in that state. Figure 1.2 shows the structure of a DQN. The neural network allows to model complex scenarios where the states space is arbitrarily huge.

1.3.3 Blockchain

Blockchain is a novel distributed data structure in which each entry, called "block", is sequentially and cryptographically linked to the previous one. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data, so that none of them can be changed. Figure 1.3 shows an example of the structure of blockchain blocks. A block body contains the transactions, the previous block hash, and a *Proof-of-Work*. A *Proof-of-Work* is a difficult mathematical problem which must be solved to add a new block to the blockchain, to deter malicious uses from adding corrupted blocks.

The blockchain is commonly built on the *distributed ledger* logic, which enables distributed transaction tracking. They are commonly managed by a peer-to-peer network, which nodes agree on a protocol to validate and add new blocks. This approach makes the blockchain suitable for economic applications, as it is owned by all the peers and does not involve a central authority able to change it. Bitcoin and Ethereum are two examples of blockchains. Ethereum also enables deployment of "Smart Contracts," which are public software programs. Once written in the blockchain, such applications cannot be modified, and are executed on all the nodes in the peer to peer network.

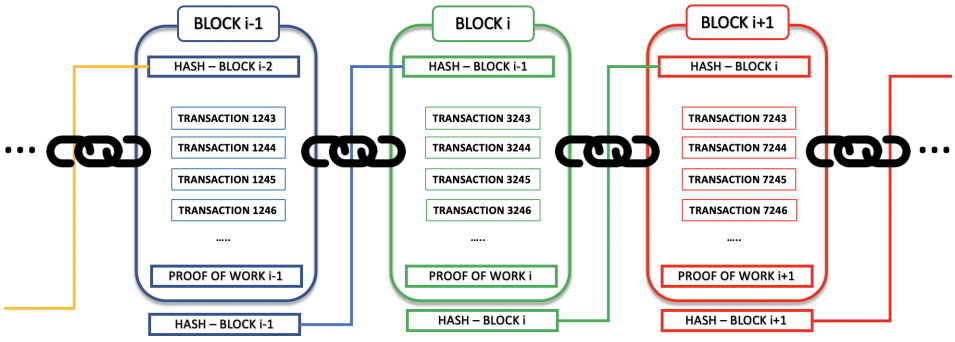


Figure 1.3. Blockchain : Block Structure Example

Chapter 2

Trajectory Planning

Fleets of cooperative drones are a powerful tool in monitoring critical scenarios requiring early anomaly discovery and intervention. Despite the large use of aerial drones in many critical settings, coordination and control is mostly left to human personnel supervising the flight operations. Automated trajectory planning is an open challenge, especially for those applicative scenarios where early inspection is of utmost importance. For instance, in critical missions where drones are looking for survivors, timeliness of intervention is a strict requirement. Thus, the benefit of using multiple aerial devices capable of sampling multiple targets in parallel, and ensuring the best trade-off between monitoring accuracy and rapidity of intervention is clear. However, planning drone trajectories is still a major challenge under several points of view, including autonomy, cooperation, coordination, and control.

In this chapter we propose new solutions to the problem of trajectory planning under safety-critical scenarios. Conversely from existing work, we let drones visit target points in consecutive trips, with recharging and data offloading in between. In fact, we specifically consider the limited energy availability of drones, i.e., their batteries are not sufficient to cover all the targets in one single trip.

In section 2.1 we design a new performance metric to capture timeliness of intervention and prioritize early coverage, namely *Weighted Progressive Coverage* (WPC). We show that WPC maximization is NP-hard, and we propose an efficient polynomial algorithm, called Greedy and Prune (GaP), with guaranteed approximation. By means of simulations we show that GaP performs close to the optimal solution and outperforms a previous approach in all the considered performance metrics, including coverage, average inspection delay, energy consumption, and computation time, in a wide range of application scenarios. Through prototype experiments we also confirm the theoretical and simulation analysis, and we demonstrate the applicability of our algorithm in real scenarios.

In Section 2.2 we analyze the problem of multi-trip coverage in a wider range of scenarios. We investigate the use of a genetic algorithm to schedule UAVs trajectories in a multi-round mission, under several objective functions. We demonstrate how the genetic approach achieves near optimal performance, considering also battery replacement/charging and data offloading operations. We show that our proposal fits various scenarios with good performance respect existing approaches.

This chapter has been extracted from the works in [18], [19] and [20].

2.1 A Multi-Trip Task Assignment for Early Target Inspection in Squads of Aerial Drones

In this section we consider the problem of monitoring a set of target points in a safety critical mission, where early inspection is of utmost importance.

Previous approaches to trajectory planning for squads of drones mainly consider a one-trip per drone assignment, with the purpose of maximizing the number of inspected targets, or minimizing the task completion time. However, as aerial drones usually have limited energy and storage availability, they may require multiple trips to provide complete monitoring coverage, with battery replacement/charging and data offloading in between. Unlike previous work we address the problem of designing the trip trajectories, so as to schedule the most inspections in the earliest trips, under the drone energy constraints. Moreover, we consider heterogeneous drones (e.g., with different batteries and energy consumption curves), that may depart from different depots.

For the purpose of evaluating and optimizing the capability of an algorithm to provide early monitoring, we define a novel metric called *Weighted Progressive Coverage* (WPC). The optimization of this metric allows to jointly address the problems of target coverage and task scheduling. The definition of WPC generalizes the classic notion of total coverage and also allows to give more value to tasks scheduled in the earlier rounds. The *accumulative coverage* metric is also introduced as a special instance of WPC, which assigns linearly decreasing values to the tasks executed in progressive rounds. We show how the optimization of the accumulative coverage metric successfully prioritizes early target inspection and minimizes the average inspection delay caused by inter-trip operations: battery replacing/recharging and/or data offloading can have a significant impact on the overall mission time.

We contribute an analytic formulation of the WPC maximization problem as an Integer Linear Programming (ILP) model and show its NP-hardness. Experiments confirm that the optimization problem has prohibitive execution times for problem instances of average to large size, requiring many hours of computations. For this reason we propose a very efficient polynomial algorithm with guaranteed approximation of the optimal solution.

We compare our proposal to several previous approaches, through extensive simulations and real field experiments, showing that our approach outperforms previous algorithms in all the relevant performance aspects including coverage, average inspection delay, as well as energy cost and computation time.

In summary, the main contributions of this work are:

- We formulate a novel performance metric, called *Weighted Progressive Coverage*, to measure the capability of a trajectory planning algorithm to provide early target inspection, and formulate the related optimization problem. We show that under appropriate settings of the weight function, weighted progressive coverage translates into traditional metrics of coverage, as well as into a new notion of accumulative coverage which clearly reflects early coverage capabilities.

- We formulate the problem of maximizing weighted progressive coverage as an ILP model and prove its NP-hardness. By means of experiments we show that such optimization has prohibitive computation times even for small problem instances, which precludes its application in safety critical scenarios, and motivates the need of polynomial time heuristics.
- We propose a novel greedy-and-prune polynomial algorithm for approximating the optimal solution of the aforementioned problem. We prove that the above algorithm has a constant factor approximation of 1/2.
- We study the proposed approaches in an extensive range of operational settings through simulations, and we confirm the analysis via real field experiments in a test-bed implementation. The study shows that the performance of our approach is close to the optimal, and considerably better than previous solutions in all the performance metrics, including coverage, inspection delay, energy consumption and computation time.

This work extends a previous conference contribution [18].

2.1.1 Related Work

Previous works addressing path planning management for teams of UAVs proposed several approaches including: graph algorithms, optimization problems, genetic and machine learning algorithms. A common requisite for all the approaches is to determine one or multiple routes satisfying given requirements related to the visit of a set of points of interest in an area.

The first approach considers variants of the Traveling Salesman Problem (TSP) for multiple vehicles [36]. Specifically, the multi-traveling salesmen problem, and its variant, the vehicle routing problem (VRP) [37], aim at finding the shortest, or lowest cost paths for the mobile devices to visit all the points of interest under a variety of constraints such as an upper bound on maximum cost for each trajectory or on the maximum number of traversed points of interest; the multi-repairmen problem aims at covering all the points of interest and minimizing the average visit delay of the points [38–40]; finally, the Capacitated Vehicle Routing Problem (CVRP) [41] aims at visiting points of interest, while delivering goods under limited vehicle capacity.

The works [16] and [17] generalize the multi-traveling salesmen problem for multi-UAV path planning for reconnaissance and surveillance contexts. However, they do not consider the limited energy availability of the drones, an inevitable constraint of any battery powered devices. In particular, in [17] the authors propose a path planning algorithm for multiple UAVs which aims at minimizing the mission completion time in search-and-reconnaissance operations. In Section 2.1.5 we design an energy-constrained variant (shortly called *TCTPA*) of this approach, which is used in our experiments for performance comparisons.

Another line of research considers the use of linear programming tools to guide decision making in mobile vehicles operations [42–44]. The work [42] proposes a novel system for wildfire monitoring using a fleet of UAVs to provide on-demand fast services, through a distributed leader-follower coalition algorithm which aims at providing full coverage of the area in a timely manner, while minimizing the drones

and the energy consumption. Despite the similarity with the problem addressed in our work, the approach proposed in [42] does not explicitly address limited resource scenarios as we do in our work to provide multiple flight rounds (with battery recharging or replacement) when the number of drones is inadequate to cover all the target points within a region in a single trip. The work in [43] studies the problem of trajectory planning for fleets of UAVs in search and rescue (SAR) operations, especially in maritime accidents. In this work the UAVs can be recharged at different stations to complete the mission. The authors propose a mixed integer linear program (MILP) to generate efficient search and rescue operation plans. The main limitation is scalability in terms of number of drones and points of interest due to the high computational complexity of the proposed solutions. The work in [44] considers a different scenario where drones are employed to provide network connectivity to a group of users on the ground. Therefore the goal of trajectory planning is to maximize users' throughput, which is different from our objective.

Other approaches leverage genetic algorithms to design the trajectories of multiple UAVs in different missions [45] [46] [47]. The work in [45] considers a fleet of drones with limited capabilities. The work solves a capacitated Vehicle Routing Problem (CVRP) for multiple drones by using a genetic algorithm providing selection, mutation and crossover. However, the solution is efficient only for scenarios with few points of interest; as the number of points to be visited increases, the required computational power increases significantly. In intelligence transportation systems, the work in [46] aims at finding time-optimal paths for multiple UAVs such that they collectively visit some target areas when they fly from a starting point to a final location. However, differently from ours, this work considers only a single-depot and a single trip trajectory for each drone.

We also mention a few solutions that are based on artificial intelligence. The work in [48] proposes an unsupervised learning algorithm for solving K-DTSP with Neighborhoods (i.e., a multi-vehicle variant of TSP where a target is considered visited provided that a drone trajectory traversed an area surrounding it) with curvature constraints. The work in [49] uses a distributed reinforcement learning algorithm to find drone paths in a dynamic scenario where the final destination for each drone is not fixed a priori. These works lack of any performance guarantee with respect to the optimal solution; moreover learning approaches usually require significant time to converge to a steady state.

Finally, there is a large number of works on mission control for ground mobile robots, which are not suitable for the study of UAV task assignment. Assumptions, requirements and constraints for UAVs are different from those for mobile robots. Aerial drones have lightweight payloads and shorter operative time, often requiring multiple trips; moreover, they consume energy also when hovering and have higher speed. The work by Setter et al. [50] addresses the problem of coordinating a squad of mobile robots to perform rendez-vous operations within device energy constraints. The work by Popescu et al. [51] tackles the robot patrolling problem to repeatedly cover a set of targets during a mission. The works by Yazici et al. [52], and Mei et al. [53] address the problem of area coverage, with homogeneous devices. Some of our experiments also study an extension of our approach to the optimization of area coverage by designing a variant of the target coverage problem. For these

experiments we also adopt the algorithm *Sweep*, inspired by [53], as a benchmark for comparisons in this problem setting.

To the best of our knowledge, our work is the first that tackles the drone path planning problem into a general scenario requiring early target inspection. We consider multiple-round exploration and maintenance phases in order to work also in under-provisioned settings (e.g., where the drone batteries are not sufficient to cover all the targets in one single trip, therefore they are allowed to return to the depot station to have their battery replaced or recharged). We propose a novel optimization problem, where the flight dynamics, energy consumption, and restricted flight zones are directly incorporated in the input and early inspection is explicitly measured and optimized; finally, we propose an efficient polynomial algorithm with guaranteed performance, evaluated against other proposals by means of both simulations and prototype experiments.

This work extends a previous conference version [18]. With respect to the previous work the major enhancements are the following: in Section 2.1.3 we introduce a novel problem formulation which directly computes trajectories and we provide a related algorithm for trajectory design in Section 10; we present real field experiments in Section 2.1.6, with an in-depth discussion on the key implementation challenges. In the Section 2.1.6 we also extend the evaluation to the different operation scenario requiring area coverage as opposed to target coverage.

2.1.2 Problem formulation

We consider the scenario in which a squad of aerial drones aims at inspecting a given set of target points Ψ in the region of interest. These points may be the known locations of survivors of a catastrophe, requiring medicines or water to be dropped from aerial drones, or more in general they may be areas of suspected anomalies, requiring immediate surveillance and local inspection. Figure 2.1 shows an example of the system with two different depots, d_1 and d_2, a set of 6 heterogeneous drones, and several critical points to visit, i.e. the red markers in the figures. The goal of our system is to compute — at the mission start — the multi-trip trajectories that the squad should follow in order to visit all points of interest.

Let $\mathcal{U}$ be the set of aerial vehicles forming the squad, and let d_u be the home depot of drone $u \in \mathcal{U}$, i.e. the point of the region of interest from which the drone departs, and where it is recollected, its data are offloaded, and batteries are recharged at the end of each trip. It is important to notice that the battery of a drone imposes a limitation on the single trip duration. For instance, the DJI AGRAS T16 [54] drone, used to nebulize water and medicines on crop fields for professional agricultural applications, can only fly for 18 minutes in hovering mode without payload, considering basic battery equipment. The flight time may be instead as low as 10 minutes when the drone is fully loaded (i.e., with a payload of about 16 Kg), an amount of time which becomes even smaller if the drone performs complex flight maneuvres. In our experimental test-bed described in Section 2.1.6, we use DJI Flame Wheel F550 [55] drones, equipped with a LIPO battery providing 3500mAh 25C, which allows the drone to fly at 5 m/s for about 7 minutes with no additional payload.

Figure 2.1. System scenario

Progressive coverage metrics

We consider a progressive execution of the monitoring activity of the drone squad in consecutive rounds $n = 1, 2, \ldots, N$, where new target points are inspected at each new round. A round based execution is meant to consider recharging and offloading at the depot between consecutive rounds. We denote with $[N]$ the set $\{1, 2, \ldots, N\}$. The variables $\delta_i(n) \in \{0, 1\}$ represent the coverage status of target i at round $n \in [N]$ for any $i \in \Psi$. If target i is visited exactly at round n, then $\delta_i(n) = 1$, otherwise $\delta_i(n) = 0$. To avoid repeated coverage of a single point i, we impose $\sum_{n=1}^{N} \delta_i(n) \leq 1$.
We denote with

$$\delta(n) \triangleq \sum_{i \in \Psi} \delta_i(n) \tag{2.1}$$

the amount of target points covered exactly at round n, that we shortly call *round coverage*.

We also define a round based utility function $w(n)$ reflecting non increasing utilities for each round, such that $w(n_1) \geq w(n_2) \geq 0$, when $n_1 < n_2$ for $n_1, n_2 \in [N]$.

We now introduce a novel metric $\mathcal{W}(n)$ called *Weighted Progressive Coverage (WPC)*, defined as the weighted average of the round coverage, calculated over the first n rounds:

$$\mathcal{W}(n) \triangleq \sum_{k=1}^{n} w(k) \cdot \delta(k). \tag{2.2}$$

The specific setting of the weights $w(k)$ at round k is meant to give different priorities to the earlier trips, considering one trip per drone at each round.

It is easy to see that the *total coverage in n rounds*, hereby denoted with $\Delta(n)$, is obtained by setting $w(k) = 1,\ \forall k = 1, \ldots, N$, in Equation (2.2) as follows:

$$\Delta(n) \triangleq \sum_{k=1}^{n} \delta(k), \tag{2.3}$$

which represents the number of targets covered either at round n or at any round before n. Hence total coverage is a special case of WPC.

We now define a novel coverage metric which prioritizes early coverage of target points, called *accumulative coverage* $\mathcal{A}(n)$ *at round* n:

$$\mathcal{A}(n) \triangleq \sum_{k=1}^{n} \Delta(k). \tag{2.4}$$

We observe that the accumulative coverage metric is also a special case of WPC, obtained by setting the values of the weights $w(k)$, $k = 1, \ldots, n$, as explained in the following observation.

Observation 2.1.1. *The accumulative coverage function* $\mathcal{A}(n)$ *at round* n *is a linear combination of the single round coverage variables which can be expressed as:*

$$\mathcal{A}(n) = \sum_{k=1}^{n} (n - k + 1) \cdot \delta(k). \tag{2.5}$$

Therefore the accumulative coverage function at round n *is a special case of WPC obtained by setting the values* $w(k) = (n - k + 1)$*, in Equation (2.2).*

Proof. By simple algebraic passages, we see that

$$\mathcal{A}(n) = \sum_{k=1}^{n} \Delta(k) = \sum_{k=1}^{n} \sum_{j=1}^{k} \delta(j) = \sum_{k=1}^{n} (n - k + 1) \cdot \delta(k). \quad \square$$

Table 2.1 summarizes the four coverage definitions.

Table 2.1. Summary of notations

Notation	Description
$\delta(n)$	*Round coverage* at round n - Eq. 2.1
$\mathcal{W}(n)$	*Weighted Progressive Coverage (WPC)* at round n - Eq. 2.2
$\Delta(n)$	*Total coverage* at round n - Eq. 2.3
$\mathcal{A}(n)$	*Accumulative coverage* at round n - Eq. 2.4

WPC and, more specifically, its accumulative coverage variant, reflects the capability of a trajectory planning algorithm to prioritize coverage in the early rounds.

As an example, let us consider a set of 30 targets to be visited within $N = 5$ rounds. The plots of Figure 2.2 show, in row order, the value of round, total, and accumulative coverage. The figure considers two different cases of task assignment, where the available drones execute different trips at each round $n \in [N]$, as evidenced by the first row, which highlights the covered targets per round, i.e., round coverage. We see that the value of total coverage, shown in the second row, is the same for the two cases at the end of the first five rounds, as both the executions ended the fifth round with complete coverage of the set of targets. Measuring the total coverage metric at the end of the mission does not capture the difference between the two

executions as it only considers what was covered at the end of the flight schedule. In contrast, the accumulative coverage value, shown in the third row, properly captures the fact that the trips designed for the second case provide higher round coverage in the early rounds. The accumulative coverage value at the end of the fifth round is in fact higher in the second case than in the first case of execution. This is due to the higher weights assigned to the initial rounds in the computation of the accumulative coverage value, which reflect the prioritization of early inspections.

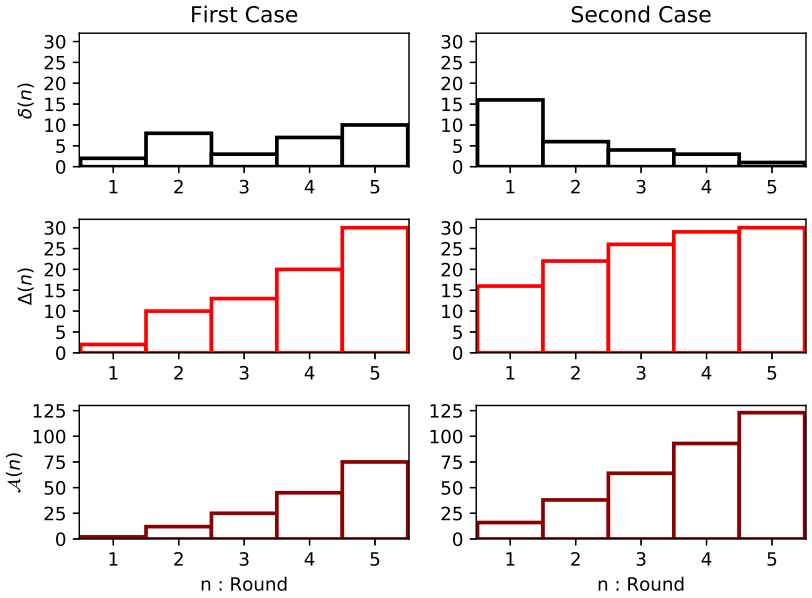


Figure 2.2. Round-, total-, and accumulative-coverage with two different choices of trajectories for multiple rounds of flight.

Consider now N rounds, and a setting where all the target points of Ψ are covered in a progressive, round based inspection, namely $\sum_{k=1}^{N} \delta(k) = |\Psi|$.

Let $\tau_i(N)$ be the round at which target i is visited, hereby called the *inspection delay* of target $i \in \Psi$ in an N-rounds progressive solution. It holds that:

$$\tau_i(N) \triangleq \sum_{k=1}^{N} k \cdot \delta_i(k). \tag{2.6}$$

We denote with $D(N)$ the *average inspection delay*:

$$D(N) = \sum_{i \in \Psi} \tau_i(N)/|\Psi|. \tag{2.7}$$

Theorem 2.1.1. *For any progressive coverage solution $\delta_i(k)$, with $i \in \Psi$ and $k \in [N]$, it holds that:*

$$\mathcal{A}(N) = |\Psi| \cdot [N + 1 - D(N)], \tag{2.8}$$

where $\mathcal{A}(N)$ and $D(N)$ are defined in Equations (2.5) and (2.7), respectively. It follows that any progressive coverage solution which guarantees complete coverage of the set Ψ and maximizes the accumulative coverage function $\mathcal{A}(N)$, also minimizes the average inspection delay $D(N)$.

Proof. By applying Observation 2.1.1 we have that

$$\mathcal{A}(N) = (N+1) \cdot \sum_{k=1}^{N} \delta(k) - \sum_{k=1}^{N} k \cdot \delta(k) =$$
$$= (N+1) \cdot |\Psi| - \sum_{k=1}^{N} \sum_{i \in \Psi} k \cdot \delta_i(k) = (N+1) \cdot |\Psi| - \sum_{i \in \Psi} \tau_i(N) =$$
$$= |\Psi| \cdot [(N+1) - D(N)] . \quad \square$$

Discussion on the setting of the number of rounds N: We clarify that the setting of N — the maximum number of rounds for inspecting the target points in the field of interest — typically responds to application requirements, for example if drones are supposed to operate during daylight hours. The setting of N affects the total duration of the target inspection mission. Assuming that each drone has enough energy to inspect at least a target of Ψ at each round, an upper bound on the setting of N is $|\Psi|/|\mathcal{U}|$, which is the maximum number of rounds required to ensure completion of the target covering mission (i.e., total coverage of the target points).

Discussion on the round length and synchronization: The WPC metric gives the same weight to the inspection of different targets when they occur at the same round. This requires that (1) the time between two consecutive rounds is at least as long as a single round duration to ensure that the time between two inspections occurring in the same round is always shorter than the time between inspections occurring in consecutive rounds; (2) inspection rounds executed by different drones have a similar length, which ensures that round based utilities are monotonically decreasing with respect to the target inspection times, regardless of the drone being considered. However we notice that the first assumption is verified in most application scenarios. In fact, maintenance operations between consecutive trips (e.g., battery recharging and data offloading) typically require a time that is a factor of 3 to 8 times the maximum flight duration. In the experimental test-bed considered in Section 2.1.6, this factor is about 10. The second assumption is also valid whenever drones have roughly homogeneous flight capabilities and their trips are terminated only when the residual energy is not sufficient to inspect any more targets, with the exception of the last round. In most operative cases, with battery powered copters, the above assumptions are verified as the inter-round operation time usually dominates the drone flight duration. If these assumptions are not verified for a specific problem instance, the WPC metric definition may be revised to consider utilities which decrease with the actual target inspection time.

2.1.3 WPC optimization

The problem of maximizing WPC or any of its variants, including total and accumulative coverage, can be formulated as an Integer Linear Programming (ILP) model, as detailed in Problem 2.1. The objective function (a) of the problem is the expression of WPC defined in Equation 2.2, where the integer variable $\delta(n)$ denotes the amount of targets covered at round n, i.e., the round coverage defined by Equation 2.1, and in constraint (h) of the problem. The binary variables $x_{ij}^{u}(n)$, with

$i, j \in \Psi \cup \{d_u\}$, $u \in \mathcal{U}$, $n \in [N]$, reflect the decision to let drone u traverse the route between the locations of target i and j, exploring them in a sequence ($x^u_{ij}(n) = 1$), or not ($x^u_{ij}(n) = 0$), at the n-th trip. Based on these variables, constraint (b) imposes that a target be traversed the same number of times in entry and exit direction. Sub-tour elimination is obtained via the MTZ technique proposed by Miller, Tucker and Zemlin in [56], with the constraints (c-d), where the auxiliary integer variables $z^u_i(n)$ represent the ordinal position of target i in the trajectory of drone u at round n. Constraint (e) imposes an upper bound b_u on the maximum energy expenditure of drone u at each round, where b_u is the drone battery availability (in energy units), and ϕ^u_{ij} is a constant value, reflecting the energy cost, for drone u to travel from target i to target j and inspect target j (we refer to Section 2.1.3 for a discussion on the setting of the parameters ϕ^u_{ij}). Constraint (f) relates the value of the target coverage variable $\delta_i(n)$ to the values of the variables $x^u_{ij}(n)$, and determines the decision to cover target $i \in \Psi$ at round n. Constraint (g) precludes redundant coverage of the same target points, constraint (h) relates the round coverage $\delta(n)$ with the values of the variables $\delta_i(n)$ denoting coverage of individual targets, while constraints (i) define the variable domains.

$$
\begin{aligned}
&\max \mathcal{W}(N) \triangleq \textstyle\sum_{i=1}^{N} w(i) \cdot \delta(i) && (a)\\
&s.t., \forall n = 1, \ldots, N \\
&\textstyle\sum_{i \in \Psi \cup \{d_u\}} x^u_{ij}(n) = \sum_{k \in \Psi \cup \{d_u\}} x^u_{jk}(n), \forall j \in \Psi, u \in \mathcal{U} && (b)\\
&z^u_j(n) - z^u_i(n) \geq x^u_{ij}(n) + |\Psi| \cdot (x^u_{ij}(n) - 1), \forall u \in \mathcal{U}, i, j \in \Psi && (c)\\
&z^u_{d_u}(n) = 0;\ z^u_i(n) \in \{1, \ldots, |\Psi|\}, \forall i \in \Psi, u \in \mathcal{U} && (d)\\
&\textstyle\sum_{ij \in \Psi \cup \{d_u\}} \phi^u_{ij} \cdot x^u_{ij}(n) \leq b_u, \forall u \in \mathcal{U} && (e)\\
&\delta_i(n) = \textstyle\sum_{u \in \mathcal{U}, j \in \Psi \cup \{d_u\}} x^u_{ij}(n), \forall i \in \Psi && (f)\\
&\textstyle\sum_{n=0}^{N} \delta_i(n) \leq 1, \forall i \in \Psi && (g)\\
&\delta(n) = \textstyle\sum_{i \in \Psi} \delta_i(n) && (h)\\
&x^u_{ij}(n) \in \{0, 1\}, \delta_i(n) \in \{0, 1\}, \forall i, j \in \Psi, u \in \mathcal{U} && (i)
\end{aligned}
$$

Problem 2.1. WPC Optimization (edge-related variables)

Formulation with restricted sets of paths

The solution space defined by the constraints of Problem 2.1 includes all the cyclic trajectories $\mathcal{C}_u$ that each drone u can traverse within its battery limitation, starting from its depot d_u. In Problem 2.2 we provide an equivalent formulation, considering trajectory related variables $z^u_p(n) \in \{0, 1\}$, to denote the decision to assign the cyclic path $p \in \mathcal{C}_u$ to drone $u \in \mathcal{U}$ at round $n \in [N]$. Constraint (b) defines the variables $\delta_i(n)$ as a function of the trajectory related variables $z^u_p(n)$, constraint (c) imposes that each drone traverses at most one of the cyclic trajectories at any given round

n, constraint (d) precludes redundant coverage, and the remaining constraints (e-f) define the domain of the decision variables of the problem.

$$\max \mathcal{W}(N) \triangleq \sum_{i=1}^{N} w(i) \cdot \delta(i) \qquad (a)$$

$$s.t.$$

$$\delta_i(n) = \sum_{u \in \mathcal{U}, p \in \mathcal{C}_u : i \in p} z_p^u(n), \forall i \in \Psi,\ \forall n \in [N] \qquad (b)$$

$$\sum_{p \in \mathcal{C}_u} z_p^u(n) \leq 1,\ \forall u \in \mathcal{U},\ \forall n \in [N] \qquad (c)$$

$$\sum_{n=1}^{N} \delta_i(n) \leq 1, \forall i \in \Psi \qquad (d)$$

$$z_p^u(n) \in \{0,1\},\ \forall u \in \mathcal{U}, p \in \mathcal{C}_u,\ \forall n \in [N] \qquad (e)$$

$$\delta_i(n) \in \{0,1\},\ \forall i \in \Psi,\ \forall n \in [N] \qquad (f)$$

Problem 2.2. WPC Optimization (path-related variables)

The problem is clearly NP-hard as it generalizes the Travelling Salesman Problem (TSP) [57]. The number of variables of Problem 2.2 grows with the number of cyclic trajectories that can be defined on Ψ. Computing all these trajectories is time-consuming, and can become the bottleneck of the entire trajectory planning problem.

Moreover, it must be noted that in practice not all the trajectories are equally viable due to no fly-zones, energy constraints, and limited maneuverability of drones [58].

Provisioning the list of viable trajectories for each drone is in itself an interesting research problem. The work by Milan et al. [59] discusses the inherent non linearity of this problem and addresses the generation of spline trajectories by means of non-linear models which take account of mechanical constraints. Trajectories may be generated by means of clustering algorithms, including density based clustering [60], agglomerate hierarchical clustering [61], and spectral clustering [62], just to mention those that may capture the multivariate nature of the target geographical distribution.

In the following we consider a set of *candidate trajectories* for each drone $\chi_u(b_u, d_u)$, as the set of trajectories that drone $u \in \mathcal{U}$ can use ($\chi_u \subseteq \mathcal{C}_u$). The definition of this set depends on the battery limitation b_u of drone u, on the adopted energy consumption model, and on the location of the related depot d_u, as well as on the other limitations discussed above. We denote $\chi \triangleq \cup_{u \in \mathcal{U}} \chi_u$.

Notice that the WPC optimization problem under restricted sets of paths χ_u, for $u \in \mathcal{U}$, requires a more general definition of the coverage variables $\delta_i(n)$ to take account of the fact that the optimization problem could select paths with overlapping coverage of some targets. This is because the use of restricted paths does not ensure the existence of a full coverage solution composed of disjoint paths only, and we cannot exclude the selection of paths traversed by different drones and/or at different rounds, containing the same targets. In Section 2.1.4 we discuss a pruning technique to be adopted to post-process a solution to the task assignment problem based on restricted paths, for removing redundantly covered target points without affecting weighted coverage.

For the purpose of redefining Problem 2.2 under a restricted set of paths χ, we generalize the definition of the variables $\delta_i(n)$ given in Section 2.1.2 to this new setting. Therefore, we define $\delta_i(n) = 1$ if n is *the first round* in which i is covered by one or more drones, and $\delta_i(n) = 0$ otherwise. The definition of $\tau_i(N)$ is generalized accordingly, so that $\tau_i(N)$ corresponds to the number of *the first round* at which target i is traversed for the first time.

Naturally, the generalized variables $\delta_i(n)$ and $\tau_i(N)$ will produce the same values described in Section 2.1.2 when adopted in a scenario where the set of paths is unrestricted.

In agreement with Equation (2.6), in which we defined $\tau_i(N)$ for the case of unrestricted path sets, we define

$$\tau_i(N) \triangleq \min\{n \in [N] : \exists(p, u) \text{ with } p \in \chi_u, i \in p, z_p^u(n) = 1\},$$

when i is covered, otherwise we set $\tau_i(N) = N + 1$.

Observation 2.1.2. *The values of $\delta_i(n),\ i \in \Psi$, are uniquely determined from the values of the variables $z_p^u(n),\ u \in \mathcal{U},\ p \in \chi_u$, defining a round based trajectory assignment. Then $\delta_i(n) = 1$ if $n = \tau_i(N)$ and $\delta_i(n) = 0$ otherwise, for $n \in [N]$.*

Problem 2.2 must be modified to take account of restricted sets of paths, and of the new variable definitions we have just exposed. The new optimization problem, under restricted sets of paths, is the following Problem 2.3, where we replaced constraint (b) of Problem 2.2 with new constraints (b1) and (b2), as follows.

$$\begin{aligned}
&\max \mathcal{W}(N) \triangleq \textstyle\sum_{i=1}^{N} w(i) \cdot \delta(i) && (a)\\
&s.t.,\ \forall n \in [N] \\
&\delta_i(n) \geq \textstyle\sum_{u \in \mathcal{U}} \sum_{p \in \chi_u : i \in p} \frac{z_p^u(n)}{|\mathcal{U}|} - N \cdot \sum_{k=1}^{n-1} \delta_i(k),\ \forall i \in \Psi && (b1)\\
&\delta_i(n) \leq 1 - \textstyle\sum_{k=1}^{n-1} \frac{\delta_i(k)}{(n-1)},\ \forall i \in \Psi && (b2)\\
&\textstyle\sum_{p \in \chi_u} z_p^u(n) \leq 1,\ \forall u \in \mathcal{U} && (c)\\
&z_p^u(n) \in \{0, 1\},\ \forall u \in \mathcal{U}, p \in \chi_u && (d)\\
&\delta_i(n) \in \{0, 1\},\ \forall i \in \Psi && (e)
\end{aligned}$$

Problem 2.3. WPC optimization (restricted sets of paths).

The metric $\delta(n) = \sum_{i \in \Psi} \delta_i(n)$ represents the *number of newly covered targets* at round n. It generalizes the definition of $\delta(n)$ given in Section 2.1.2 for the case of unrestricted sets of paths.

Energy Models

Problem 2.1 abstracts the details of the energy model of a drone u by introducing the parameters related to the edge cost ϕ_{ij}^u, which include the costs of travelling from target i to target j, and the inspection cost at target j.

With a similar approach, Problems 2.2 and 2.3 also incorporate the energy limitation b_u of a drone u into the definition of the set of feasible trajectories $\mathcal{C}_u$ and χ_u, i.e., trajectories that each drone can traverse without depleting its battery before the end of trip.

By making the assumption that energy consumption is only related to inspection tasks and trajectories, this approach neglects environmental conditions, e.g., varying wind conditions. However, in both the formulations, the abstraction has the advantage of incorporating the energy model in linear terms of the ILP formulation. Many different energy models can be used to parametrize our problem formulation, e.g., those proposed by Goss et al. [63], while still keeping the optimization model simple and linear.

Moreover, the proposed approach has the advantage of considering drone heterogeneity, as both the parameters ϕ_{ij}^u of Problem 2.1 and the setting of feasible trajectories $\mathcal{C}_u$ and χ_u of Problems 2.2 and 2.3, respectively, can be calculated on the basis of device specific energy models.

Since no unique energy model can be general enough to capture the peculiarities of any specific device, in our experiments we adopt a simple linear model for which energy consumption is proportional to the travelled distance when the drone moves, and to the flight time when the drone hovers to inspect a target. This model is discussed in Section 2.1.5.

2.1.4 Efficient approximation algorithms for WPC optimization

The definition of restricted sets of paths χ_u for each drone $u \in \mathcal{U}$ significantly reduces the size of the feasible region of the WPC maximization problem, i.e., it reduces the number of cycles to be considered as potential trajectories. Theorem 2.1.2 shows that, despite these simplifications, the problem is still NP-hard, and motivates research for polynomial time heuristics with good approximation of the optimal, within the constrained sets of trajectories.

Theorem 2.1.2. *Problem (2.3) is NP-hard.*

Proof. Any instance of the Max-Coverage problem can be reduced to an instance of Problem 2.3 in polynomial time. Consider a collection $\mathcal{S}$ of sets of elements of $\mathcal{T}$, and an integer value m. The Max-Coverage problem requires finding m elements $S_1, S_2, \ldots, S_m$ of $\mathcal{S}$ such that $\cup_{i=1}^{m} S_i$ is maximized. We can build an instance of our WPC problem by considering all the elements of $\mathcal{T}$, as target points, therefore $\Psi = \mathcal{T}$, and one only drone u, $|\mathcal{U}| = 1$ with depot d_u, flying for $N = m$ rounds. The restricted set of paths χ_u associated to drone u is then formulated as follows: for any $S \in \mathcal{S}$ we consider a path $p_S \in \chi_u$ including all the nodes of S and the depot d_u in a cyclic trajectory (any cycle fits our needs). We then set the energy availability of drone u to b_u such that b_u is sufficient to complete any trajectory p_S for any $S \in \mathcal{S}$. Finally, we set the weights $w(n) = 1$, for each $n = 1, \ldots, m$. Any solution to Problem 2.3 maximizes the number of elements of $\mathcal{T}$ covered by m sets of $\mathcal{S}$. The hardness of Problem 2.3 derives from the hardness of Max-Coverage. □

The hardness of Problem 2.3 motivates us to seek for efficient suboptimal solutions with guaranteed performance. To this end, we identify properties of the set

of constraints and of the objective functions that allow for easy approximation. In the following we assume that the sets of feasible trajectories χ_u is known in advance, for each drone $u \in \mathcal{U}$, and defines the problem instance. In Section 10 we discuss efficient techniques for generating sets of feasible trajectories.

Trajectory Assignment as Matroid Optimization

We recall the following concepts from combinatorial optimization.

Definition 2.1.1 (Matroid [64]). *A matroid $\mathbb{M}$ is a pair (E, I), where E is a finite ground set and $I \subseteq 2^E$ a non-empty collection of subsets of E, with the following properties:*

1. *$\forall A \subset B \subseteq E$, if $B \in I$, then $A \in I$;*
2. *$\forall A, B \in I$ with $|B| > |A|$, $\exists x \in B \setminus A$ such that $A \cup \{x\} \in I$.*

Definition 2.1.2 (Partition Matroid [65]). *A matroid $\mathbb{M} = (E, I)$ is a* partition matroid *if E is partitioned into* disjoint *sets $E_1, E_2, \ldots, E_m$ and, for some given integers $b_1, \ldots, b_m$, $0 \leq b_i \leq |E_i|$,*

$$I = \{X \subseteq E \ : \ |E_i \cap X| \leq b_i, \ for \ i = 1, 2, \ldots, m\}.$$

Definition 2.1.3 (Monotone submodular function [64]). *Given a finite ground set E and a function $f : 2^E \to \mathbb{R}$,*

- *f is monotone if $\forall A \subset B \subseteq E$, $f(A) \leq f(B)$;*
- *f is submodular if $\forall A \subset B \subseteq E$ and $e \in E \setminus B$, $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$.*

By proving that our objective function is monotonically increasing and submodular, and that our constraints form a matroid, we can apply results from combinatorial optimization to define algorithms with known approximation guarantee. More specifically, we recall the following result on the greedy approach[1].

Theorem 2.1.3 ([67]). *Consider the maximization of a set function $f : 2^E \to \mathbb{R}$ over a collection $I \subseteq 2^E$ of sets. We denote with f^* the optimal value, and with f^g the value achieved by the greedy algorithm. If $\mathbb{M} = (E, I)$ is a matroid and f is monotone and submodular, then $f^g \geq f^*/2$.*

We show that our trajectory assignment problem can be cast as the problem of maximizing a set function under matroid constraints. In fact, under the assumption that each drone u may be activated a given number of rounds at most equal to k_u, with full battery charge b_u, the constraints of Problem 2.3 define a set of feasible solutions that can be mapped to a partition matroid.

Each drone selects up to k_u cycles on a round basis, for a total of maximum N rounds, where $N = \max_{u \in \mathcal{U}} k_u$. For simplicity, we hereby consider $k_u = N, \ \forall u \in \mathcal{U}$. Notice that the round-based cycle selection does not imply that the drone activities

[1] A more complex algorithm with the best approximation known so far, $(1 - 1/e)$, was proposed by Calinescu et al. in [66].

must be synchronized. Nevertheless as a first study we consider the round based weight decreasing in the number of elapsed rounds (namely, targets visited in the first round have the largest weight, regardless of the time taken by their drone to complete the round).

In the following lemma we show that the feasible region of Problem 2.3 can be mapped to a matroid.

Lemma 2.1.1. *The solution space of Problem 2.3 can be modelled by means of the pair $M = (\mathcal{E}, \mathcal{I})$, defined as follows:*

- *$\mathcal{E}$ is a ground set, $\mathcal{E} = [N] \times \chi$ composed of all the cyclic trajectories for any vehicle $u \in \mathcal{U}$, each cloned N times, one for each round: $\mathcal{E} = \{(i, p) : i \in [N], p \in \chi\}$. Such a ground set is partitioned into $N \cdot |\mathcal{U}|$ disjoint subsets $\mathcal{E}_{i,u} = \{(i, p) : \ p \in \chi_u\}, \ \forall i \in [N]$ and $u \in \mathcal{U}$.*
- *$\mathcal{I}$ is nonempty collection of subsets of $\mathcal{E}$ defined as $\mathcal{I} = \{S \subseteq \mathcal{E} \ : \ |S \cap \mathcal{E}_{j,u}| \leq 1, \forall (j, u) \in [N] \times \mathcal{U}\}$.*

The pair $M = (\mathcal{E}, \mathcal{I})$ is a matroid (a partition matroid).

Proof. We first establish a one to one mapping between any feasible solution of Problem 2.3, identified by the variable vectors $\bar{\delta}$ and $\bar{z}$, and a set $S = S(\bar{\delta}, \bar{z}) \in \mathcal{I}$, then we show that M is a matroid.
(One to one mapping.) Given a solution of Problem 2.3, we can build the corresponding set $S \in \mathcal{I}$ consisting of all the elements (n, p), with $p \in \chi_u$, for which $z_p^u(n) = 1$. Thanks to constraint (c) of Problem 2.3, there will be at most one element in $S \cap \mathcal{E}_{n,u}$, for each $n \in [N]$ and $u \in \mathcal{U}$. The opposite is also true. If we take any set $S \in \mathcal{I}$, this will have at most one element in each partition subset $\mathcal{E}_{n,u}$ of the matroid M. More specifically, if $S \cap \mathcal{E}_{n,u} = \emptyset$, no trajectory of χ_u is assigned to drone u at round n and $z_p^u(n) = 0$, for all $p \in \chi_u$ and for the selected drone $u \in U$ at round n. If instead the set $S \cap \mathcal{E}_{n,u}$ is not empty, by the definition of the partition matroid M, S will contain only one element (n, p^*) with $p^* \in \chi_u$. This corresponds to assigning $z_{p^*}^u(n) = 1$, and $z_p^u(n) = 0, \ \forall p \neq p^*$. The variables $\bar{\delta}$ are defined according to the setting of $\bar{z}$ in agreement with constraints (b1-b2) of Problem 2.3, or equivalently using the definition introduced in Observation 2.1.2.
(M is a matroid.) In order to prove that the pair $M = (\mathcal{E}, \mathcal{I})$ is actually a matroid, we must verify the two conditions of Definition 2.1.1 on M. For condition (1), let us consider A and B, with $A \subset B \subseteq \mathcal{E}$. A and B are sets of indexed cyclic trajectories, for selected vehicles and round indexes. As a consequence, if B belongs to the solution space $\mathcal{I}$ then A also belongs to $\mathcal{I}$ (A represents a solution with fewer trajectories than in solution B). For condition (2), given any A and B in $\mathcal{I}$, such that $|B| > |A|$, we can write $|B| = \sum_{(i,u) \in [N] \times U} |B \cap \mathcal{E}_{i,u}|$ and $|A| = \sum_{(i,u) \in [N] \times U} |A \cap \mathcal{E}_{i,u}|$, because the sets $\mathcal{E}_{i,u}$ constitute a partition of $\mathcal{E}$.

We can see that there exists at least an element $x = (i', u') \in [N] \times \mathcal{U}$ such that $|B \cap \mathcal{E}_{i',u'}| = 1$ and $|A \cap \mathcal{E}_{i',u'}| = 0$. Therefore the element x that satisfies the second condition of Definition 2.1.1 is such that $\{x\} = B \cap \mathcal{E}_{i',u'}$. Indeed, $x \in B \setminus A$ and $A \cup \{x\}$ is still a solution in $\mathcal{I}$, i.e., $A \cup \{x\} \in \mathcal{I}$. It is straightforward to verify that the definition of the matroid M is consistent with Definition 2.1.2, hence M is a partition matroid. □

In the following, with p we interchangeably denote either the cycle or the set of nodes traversed by the cycle.

Lemma 2.1.2. *Problem 2.3 can be cast as a set function optimization on the partition matroid M defined in Lemma 2.1.1, where the objective function is*

$$\tilde{f}(S) \triangleq \sum_{n=1}^{N} w(n) \cdot |\cup_{(n,p)\in S} p \setminus \cup_{(i,p)\in S: i<n} p|. \tag{2.9}$$

Proof. We consider the objective function of Problem 2.3 and express it in terms of the elements of matroid M. We recall that the objective function of Problem 2.3 is $\mathcal{W}(N) = \sum_{n=1}^{N} w(n) \cdot \delta(n)$. According to Observation 2.1.2, $\delta_i(n) = 1$ if $n = \tau_i(N)$ and $\delta_i(n) = 0$ otherwise, for $n \in [N]$. Therefore the value of $\delta(n)$ is the cardinality of the set of targets visited at round n for the first time. Consequently, in terms of the matroid M we can write

$$\delta(n) = |\cup_{(n,p)\in S} p \setminus \cup_{(i,p)\in S: i<n} p|. \tag{2.10}$$

It follows that $\tilde{f}(S) = \sum_{n=1}^{N} w(n) \cdot \delta(n) = \mathcal{W}(N)$. □

Lemmas 2.1.1 and 2.1.2 show that the WPC problem can be cast as the optimization of a set function over a matroid constraint. In the following we prove that this set function is monotone submodular.

Theorem 2.1.4. *Function $\tilde{f} : \mathcal{I} \to \mathbb{R}$ is a monotone submodular function on the matroid $M = (\mathcal{E}, \mathcal{I})$.*

Proof. We discuss monotonicity and submodularity separately.
Monotonicity proof: Let us consider $A, B \in \mathcal{I}$ such that $A \subset B$. We must show that $\tilde{f}(B) \geq \tilde{f}(A)$.

Given a solution S we define with $\tilde{\tau}_S(i)$ the value of

$$\tilde{\tau}_S(i) \triangleq \min\{n \in [N] : \exists (n,p) \in S \text{ with } p \in \chi, i \in p\},$$

if i is covered, and $\tilde{\tau}_S(i) = N + 1$ otherwise. Notice that, given Lemma 2.1.2, $\tilde{f}(S)$ can be rewritten as

$$\tilde{f}(S) = \sum_{i\in\Psi} w(\tilde{\tau}_S(i)) \tag{2.11}$$

Since $A \subset B$, it holds $\tilde{\tau}_A(i) \geq \tilde{\tau}_B(i),\ \forall i \in \Psi$. As a consequence $w(\tilde{\tau}_A(i)) \leq w(\tilde{\tau}_B(i)),\ \forall i \in \Psi$, because $w(n)$ is a non-increasing function of n. Therefore, applying Equation (2.11), we obtain

$$\tilde{f}(B) = \sum_{i\in\Psi} w(\tilde{\tau}_B(i)) \geq \sum_{i\in\Psi} w(\tilde{\tau}_A(i)) = \tilde{f}(A).$$

Submodularity proof:
To prove the submodularity of $\tilde{f}$, according to Definition 2.1.3 we must prove that $\forall A \subset B \subseteq \mathcal{E}$ and $e \in \mathcal{E} \setminus B$, $\tilde{f}(A \cup \{e\}) - \tilde{f}(A) \geq \tilde{f}(B \cup \{e\}) - \tilde{f}(B)$.

Let $e = (n_e, p_e)$ be the generic element of $\mathcal{E} \setminus B$. For a generic solution S, applying Equation (2.11), we obtain

$$\tilde{f}(S \cup \{e\}) - \tilde{f}(S) = \sum_{i \in p_e} \max\{w(n_e) - w(\tilde{\tau}_S(i));\ 0\}.$$

As we observed for monotonicity, $A \subset B$ implies $\tilde{\tau}_A(i) \geq \tilde{\tau}_B(i)$, and consequently $w(\tilde{\tau}_A(i)) \leq w(\tilde{\tau}_B(i))$, $\forall i \in \Psi$, as we recall that $w(n)$ is non-increasing in n. Therefore $\tilde{f}(A \cup \{e\}) - \tilde{f}(A) = \sum_{i \in p_e} \max\{w(n_e) - w(\tilde{\tau}_A(i)), 0\} \geq \sum_{i \in p_e} \max\{w(n_e) - w(\tilde{\tau}_B(i)), 0\} = \tilde{f}(B \cup \{e\}) - \tilde{f}(B)$, which proves submodularity.

□

As a consequence of this observation we can apply Theorem 2.1.3 to define a greedy approach with 1/2-approximation of the optimal.

Algorithm Greedy and Prune (GaP) for weighted progressive coverage

In agreement with the discussion of Section 2.1.4 on the constant factor approximation of the greedy approach, we propose the adoption of an enhanced greedy algorithm, called Greedy and Prune (GaP) described in Algorithm 1. This algorithm provides a preliminary greedy trajectory assignment phase, followed by a pruning step which removes redundant target points from the selected trajectories, without affecting WPC .

Algorithm 1: Greedy and Prune (GaP)

Input: A set of drones $\mathcal{U}$, a family of candidate cyclic trajectories $\{\chi_u : u \in \mathcal{U}\}$, a number of rounds N, and a weight function $w : [N] \to \mathbb{R}$

Output: An assignment of $|\mathcal{U}|$ ordered lists of cycles $\bar{\mathcal{L}}$, where $\mathcal{L}_u = (p_1^u, \ldots, p_N^u) \in \chi_u$ to each drone $u \in \mathcal{U}$

$\mathcal{R}_{\texttt{assigned}} = (r_1, r_2, \ldots, r_{|\mathcal{U}|}) \leftarrow (1, 1, \ldots, 1)$
$\mathcal{V} \leftarrow \emptyset$, $\bar{\mathcal{L}} \leftarrow \emptyset^{|U|}$
while $\exists u \in \mathcal{U}$ *s.t.* $r_u \leq N \wedge \cup_{p \in \chi_u} p \setminus \mathcal{V} \neq \emptyset$ **do**
  $(u^*, p^*) = \arg\max_{u \in \mathcal{U}, p \in \chi_u:\ r_u \leq N} w(r_u) \cdot |p \setminus \mathcal{V}|$
  $\chi_{u^*} = \chi_{u^*} \setminus p^*$
  $r_{u^*} = r_{u^*} + 1$
  $\mathcal{V} = \mathcal{V} \cup \{p^* \cap \Psi\}$
  append p_{u^*} to $\mathcal{L}_{u^*}$
prune of redundantly covered targets
return $\bar{\mathcal{L}}$

GaP iteratively builds an assignment of paths for the drones, along the available rounds, by selecting at each step the next tour that maximizes the WPC . When the set of feasible paths χ is restricted, the solution may contain overlapping trajectories, namely trajectories assigned to distinct drones or to the same drone at distinct rounds, containing at least a common target point. In such cases, the pruning algorithm removes this redundancy without affecting WPC. In particular, it removes

a target t from a path p if exist an earlier round with a path p' s.t. $t \in p'$. If overlaps occur in the same round, the algorithm prune every occurrence but one. Clearly, as the algorithm removes only redundant targets and leaves them in earliest rounds, it provides the same value of WPC as a pure greedy approach. Therefore we derive the following corollary.

Corollary 2.1.1. *GaP achieves* $1/2$*-approximation of the optimal solution to the* WPC maximization problem *under restricted sets of trajectories*[2].

That is, the WPC obtained by the trajectories selected by GaP is at least half of the maximum that can be obtained by optimally choosing the drone trajectories within the restricted feasible sets χ_u.

We recall that the formulation of weighted progressive coverage generalizes the other coverage metrics described in Section 2.1.2. The total coverage metric of Equation (2.3) and the accumulative coverage metric of Equation (2.5) can be obtained by setting the weight parameters $w(k)$, $k = 1, \ldots, N$, to specific values, such that $w(k)$ is non increasing with k. Therefore Theorem 2.1.4 is still valid for assessing monotonicity and submodularity of the total and of the accumulative coverage metric, which implies that the Greedy and Prune approach of Algorithm 1 provides a constant factor approximation of $1/2$ also for these metrics. The result naturally extends to the case of unrestricted sets of trajectories, considering $\mathcal{C}_u = \chi_u$.

Theorem 2.1.5. *The complexity of GaP is* $\mathcal{O}(|\mathcal{U}| \cdot |\mathcal{X}| \cdot |\Psi| \cdot N)$

Proof. The greedy assignment of trajectories runs in $\mathcal{O}(|\mathcal{U}| \cdot |\mathcal{X}| \cdot |\Psi| \cdot N)$. In fact, it iterates in the while loop (lines 3-8) at most $N \cdot |\mathcal{U}|$ times, because at each iteration at the algorithm selects a new path for a drone u, and the related drone index r_u is increased. Its body loop can easily be implemented to run in $\mathcal{O}(|\mathcal{X}| \cdot |\Psi|)$, which is the time of dominant instruction at line 4. In fact, assuming that the weighted function $w(x)$ is constant ($\mathcal{O}(1)$), the $\arg\max$ function runs in $\mathcal{O}(|\mathcal{X}| \cdot |\Psi|)$ while the other instructions in $\mathcal{O}(|\Psi|)$.

The pruning step can be implemented in polynomial time in $\mathcal{O}(|\mathcal{U}| \cdot |\Psi| \cdot N)$. In fact, the pruning step iterates over the trajectories selected in the while loop, and updates a list of visited targets, sorted by round number, so that it can remove redundant targets already in the list, and shortcut the related trajectories. □

Generation of the restricted sets of paths

For the formulation of Problem 2.3, we have assumed that a set of feasible drone trajectories is part of the definition of the WPC optimization problem instance. In the following we discuss a technique to compute a set of efficient candidate trajectories, taking account of the energy limitations of the drones. Algorithm 2 is used to generate a subset of feasible trajectories for our simulations in Section 2.1.5 and prototype experiments in Section 2.1.6.

The algorithm initially calculates an approximate solution of TSP over the set of targets points $\Psi \cup \{d_u\}$, for each drone $u \in \mathcal{U}$. This step is based on the 1.5-approximation algorithm for TSP proposed by Christofides [15] which produces a

[2] A relaxation to the continuous and consequent random rounding of the problem 2.3 conducted with the technique described in [66] guarantees an approximation factor of $1 - 1/e$.

Algorithm 2: Drone-trajectory generation

input : drones $\mathcal{U}$, targets Ψ, energy consumption models $\omega_u : \mathcal{C}_u, \forall u \in \mathcal{U} \rightarrow \mathbb{R}$
output : Candidate trajectories $\{\chi_u : u \in \mathcal{U}\}$

```
1  sol= ∅;
2  for u ∈ U do
3  |   χ_u = ∅;
4  |   τ_u = 1.5-TSP(Ψ ∪ {d_u});
5  |   sort(τ_u); //τ_u =< d_u, v_1, v_2, ..., v_|Ψ| >;
6  |   for i ∈ {1, ..., |Ψ| − 1} do
7  |   |   for j ∈ {i + 1, ..., |Ψ|} do
8  |   |   |   τ'_u =< d_u, v_i, ..., v_j, d_u >;
9  |   |   |   if ω_u(τ'_u) ≤ b_u then
10 |   |   |   |   χ_u = χ_u ∪ {τ'_u};
11 |   sol = sol ∪ {χ_u};
12 return sol
```

route τ_u. The target points are then numbered according to their order of visit in τ_u, starting from the depot: $\tau_u =< d_u, v_1, v_2, \ldots, v_{|\Psi|} >$. The algorithm computes the restricted set of trajectories for drone u, by extracting all the possible sub-tours from τ_u which, connected to the depot d_u, meet the energy limitation b_u, i.e. all the sub-tours τ'_u for which $\omega_u(\tau'_u) \leq b_u$, where $\omega_u : \mathcal{C}_u \rightarrow \mathbb{R}$ gives the energy consumption of traversing τ_u according to the energy model of drone u.

As the drones may have different depots, energy availability and energy consumption models, the algorithm iterates on each of them to produce different sets of trajectories χ_u, $\forall u \in \mathcal{U}$, generating at most $|\Psi| \cdot (|\Psi| - 1)/2$ candidate trajectories for each drone. If the energy model allows polynomial time verification of the feasibility of a trajectory, Algorithm 2 is also polynomial in the number of drones $|\mathcal{U}|$ and targets $|\Psi|$.

2.1.5 Performance evaluation

In the following we give a simulative evaluation of the algorithms discussed in this work, while we devote Section 2.1.6 to real prototype experiments. We recall that both Problem 2.3 and the GaP algorithm aim at optimizing weighted progressive coverage, in its general formulation given in Equation (2.2), for any setting of the weight function $w(k)$, $k = 1, \ldots, N$. In this section we consider the two practical coverage metrics introduced in Section 2.1.2, namely *total coverage* $\Delta(N)$ defined by Equation (2.3), and *accumulative coverage* $\mathcal{A}(N)$ defined by Equation (2.5). Total and accumulative coverage are instances of WPC obtained by setting $w(k) = 1$ for total coverage in N rounds, and $w(k) = N - k + 1$ for accumulative coverage in N rounds, $\forall k = 1, \ldots, N$.

We hereby refer to the two variants of the optimal solution with the names TC-OPT and AC-OPT, for total coverage and accumulative coverage optimization, respectively. Similarly, the GaP algorithm variants corresponding to the two objectives are called TC-GaP and AC-GaP.

In the following, we compare the aforementioned approaches against a previous proposal, introduced by Kim et al. [17], hereby referred to with the name Tree Cover Trajectory Planning Algorithm (TCTPA) as it is based on the creation of tree covers rooted at the depot locations. In Section 2.1.5 we discuss this previous approach, explaining its limitations when applied to the scenario considered in this work, and showing the way in which they are addressed.

Energy model and restricted set of paths

While our algorithms work independently of the specific technique adopted to generate the restricted set of paths χ_u for each drone, in this experimental section we adopt the drone trajectories generated by Algorithm 2, considering that the energy consumed by a drone is proportional to the traversed distance and length of hovering time.

Concerning hovering, if ϕ_i is the necessary time to inspect the target point i, the energy consumption related to target inspection is assumed to be proportional to ϕ_i. We also assume $\phi_{d_u} = 0$ for each depot d_u of any drone $u \in \mathcal{U}$.

Given any two points $i, j \in \Psi$ we denote with ℓ_{ij} the distance that a drone needs to traverse to move from point i to point j. With an abuse of notation we say that $i \in p$ if target point i is traversed by p and that $(i, j) \in p$ if the target points i, j are traversed by p in a sequence.

The energy expenditure for traversing a path p, including hovering on the traversed target points, is $E(p) \triangleq a \cdot \sum_{i \in p} \phi_i + b \cdot \sum_{(i,j) \in p} \ell_{ij}$, where a and b are dimensional coefficients which reflect the energy consumption in energy units (eu) for a second of inspection of a target and a meter of flight, respectively. In all the experiments we consider $a = 1\text{eu/sec}$ and $b = 1\text{eu/m}$.

Tree Cover Trajectory Planning Algorithm (TCTPA)

In [17] the authors address the problem of planning multiple drone trajectories to inspect a number of target points positioned in an area of interest. The primary goal is to ensure complete coverage of the target points, while minimizing the mission completion time, i.e. the time at which the last drone returns to its depot.

TCTPA builds a graph whose vertices coincide with the target points and depots. Then, it builds a set of $|\mathcal{U}|$ balanced tree covers rooted at the depot stations, and convert each tree to a drone trajectory by means of a technique inspired to the Christofides algorithm for the TSP [15].

The authors consider a multiple depot scenario, but assume each device has unlimited energy, therefore the drone squad is always able to inspect all the targets in a unique round. Unlike this previous approach, our algorithms work in a more general scenario, where the flight autonomy of each vehicle is constrained because of the limited energy availability of each drone. To have fair comparisons, we made TCTPA work under limited energy availability, letting drones fly in multiple rounds N, possibly depleting their batteries before the completion of the target inspection mission. To make TCTPA produce N tours for each drone, we let it work as if the number of available drones was $N \cdot |\mathcal{U}|$, considering N virtual clones for each

physical drone, each with its logically replicated depot, and we let each physical drone traverse one of its clones' tours at each round.

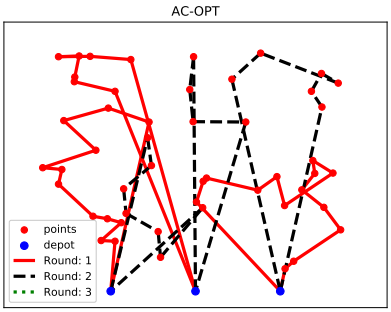


Figure 2.3. Consecutive flights under AC-OPT.

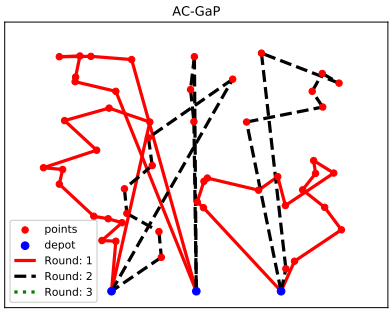


Figure 2.4. Consecutive flights under AC-GaP.

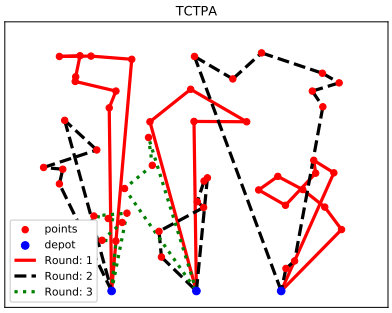


Figure 2.5. Consecutive flights under TCTPA.

Figures 2.3, 2.4, and 2.5, show the execution of AC-OPT, AC-GaP, and TCTPA, respectively, in a field of interest of 600m $\times$600m. Three drones are launched from different depots located on the bottom border of the field, and are required to inspect 50 target points randomly generated in the field area. Each drone speed is 8m/s, the drone battery is set to allow a continuous flight of about 2000m in a single round. The inspection of each target requires 5 sec of hovering time.

The figures show that AC-OPT and AC-GaP definitely succeed in designing trajectories which, under the battery limitation, contain more target points in the early rounds. The first round trajectories produced by AC-OPT and AC-GaP for the three drones are almost the same and contain a number of targets which is much higher than the average number of targets in the successive rounds. Both these algorithms complete the mission of inspecting the 50 targets in just two rounds. By contrast, TCTPA produces a more uniform distribution of targets in the exploration time, and ends up requiring one additional round for the two drones on the left of the figure.

These figures show that, in the addressed scenario, TCTPA performs worse than AC-OPT and AC-GaP in terms of accumulative coverage as it does not give priority to target coverage in the early rounds. They also show that TCTPA requires a longer completion time (in number of rounds) than the other two algorithms, which is its specific objective. Notice that in emergency critical settings, the time

interval between consecutive rounds is devoted to recharging, maintenance and data offloading, which are time consuming activities. Hence a solution that minimizes the number of rounds, together with early coverage, such as AC-GaP, is preferable in these settings.

Performance comparisons

In this section we give performance comparisons through simulations, evaluating several key performance metrics under different settings. Where not otherwise stated, the experiments consider a drone fleet of 5 vehicles, monitoring a squared area of interest of 2000m $\times$2000m with randomly located target points requiring 5sec of hovering time, with depots uniformly deployed at 100m of distance, out of the area of interest, to reflect the inaccessibility of the monitored field. Each of the drones executes a maximum number of $N = 20$ rounds of flight, with a speed of 8m/sec, under a uniform battery constraint b for each drone, which allows a flight of 7200m at constant speed (for 15min). Each point in the plots is the result of 30 runs, and the error bars denote one standard deviation of uncertainty.

Progressive coverage percentage

In the experiments we will evaluate the algorithms in terms of several performance metrics including the achieved *progressive total coverage percentage*, hereby defined as the percentage of target points covered up to a given round (according to Equation (2.3)), evaluated in progressive rounds. A progressive coverage percentage of $x\%$ at round n reflects a situation in which $x\%$ of the target points have been covered in any round from the first to the n-th.

Fig. 2.6 compares the two optimal and the two greedy approaches with TCTPA, in a setting with 225 target points, in terms of progressive total coverage achieved round by round. It is worth noting that the area under the plot until round n is equal to the accumulative coverage at round n divided by the total number of target points. The figure shows that, after AC-OPT, the algorithm that performs the best, both in total coverage and in accumulative coverage, is AC-GaP, with an excellent approximation of the optimal. Both the algorithms achieve very high coverage in the

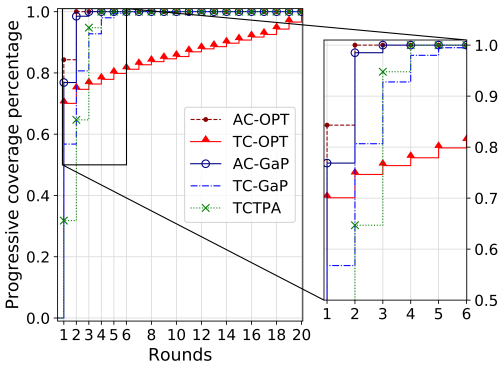


Figure 2.6. Progressive coverage (225 target points).

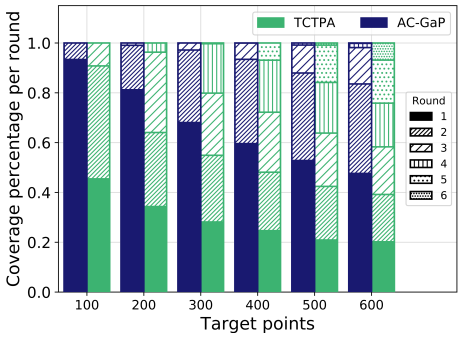


Figure 2.7. Coverage percentage (varying nr. of targets).

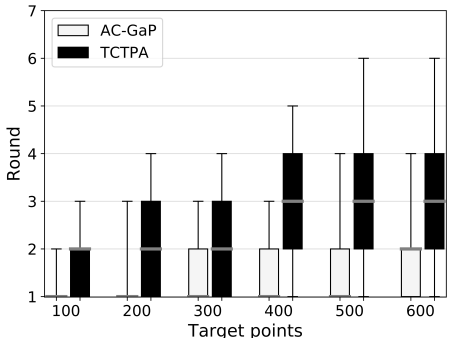


Figure 2.8. Quartiles of progressive coverage.

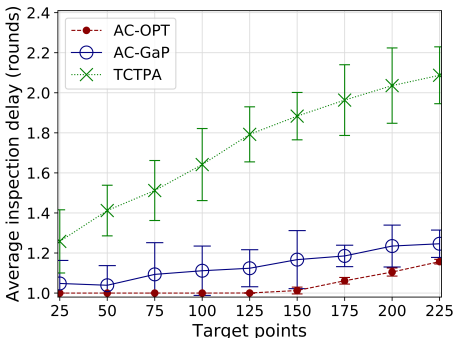


Figure 2.9. Average inspection delay in rounds.

early rounds of execution because they adopt decreasing weights $w(n)$ as prescribed by Equation 2.2 to give decreasing priority to the coverage achieved at any round $n = 1, \ldots, N$.

It is interesting to notice that in terms of progressive total coverage, TC-GaP performs better than TC-OPT, until round 20 in which both algorithms achieve 100% of total coverage. This is due to the fact that by setting the number of rounds to $N = 20$, and the weight parameters to $w(n) = 1$, TC-GaP and TC-OPT make any of the 20 rounds as important as the others in terms of coverage. None of these algorithms aim at obtaining high coverage in the early rounds, but only at the 20-th round. An algorithm that obtains zero coverage at the first 19 rounds and obtains complete coverage at round 20, has the same value of total coverage $\Delta(N = 20)$ as another algorithm that obtains complete coverage since the very first round. Finally, we observe that TCTPA has a very good performance in this setting in terms of total coverage after round 3. Nevertheless in the early rounds TCTPA performs poorly with respect to all the other algorithms, and in particular, in the first round it achieves only less than half the coverage obtained by our AC-GaP.

Figure 2.7 shows the coverage percentage achieved at any round by AC-GaP and TCTPA in settings with a number of target points growing from 100 to 650. The stacked bars show the percentage of points covered at each round (i.e., from the points covered at the first round, corresponding to the stack filled with solid texture, up to the last used round). For example, when the number of target points is 100, AC-GaP covers more than 90% of the targets within the first round (solid color block), and the remaining targets in the second round (block with dense diagonal lines). In contrast, AC-GaP covers only a little more than the 40% of the targets within the first round, and a little more than the 90% by the end of the second round, although it also needs a third round to complete target coverage. Due to the large size of the problem instances, we do not include the optimal policies in this figure, nor do we include the TC-GaP heuristics because its purpose is not to obtain high coverage in the early rounds but only within a given round N. The figure shows that also in this setting, for any round $n = 1, \ldots, N$ the coverage obtained by AC-GaP is always higher than with TCTPA. These results match the analysis of Figure 2.8 where we use a box-with-whiskers plot to show, by varying the number

of target points, the round at which the algorithms achieve the different coverage quartiles. The figure highlights that AC-GaP meets all the quartiles in an earlier round than TCTPA, demonstrating the superiority of AC-GaP in total coverage and in accumulative coverage per round.

Average inspection delay

The average inspection delay, in number of rounds, has been introduced with Equation 2.7. With Theorem 2.1.1 we proved that AC-OPT is optimal both for accumulative coverage and for average inspection delay in rounds. Figures 2.9 compares our algorithms in terms of this metric, for a growing number of target points and confirms the optimality of AC-OPT. Due to the high computational time of the optimal solution (discussed in Section 2.1.5), we only compare the optimal AC-OPT with the heuristic algorithms AC-GaP and TCTPA for small problem instances in Figure 2.9, whereas when the size of the problem instance grows, as shown in Figure 2.10, we compare the two heuristics AC-GaP and TCTPA alone. As explained by Theorem 2.1.1, AC-OPT performs always better than the other algorithms in terms of average inspection delay in rounds. Indeed, Figures 2.9 and 2.10 show that the inspection delay in rounds of algorithm TCTPA diverges from the one of AC-OPT and AC-GaP, showing that the latter algorithms excel in achieving early monitoring of target points. Figure 2.11 shows a different experiment in which we still consider the average inspection delay, but we measure it in seconds, by considering zero waiting time between two consecutive rounds, in order not to penalize TCTPA with this setting. The figure shows that also in these terms AC-GaP performs better than TCTPA, due to the fact that TCTPA prioritizes the completion time of the algorithm, which is the inspection time of the last target point, at the expense of an increased inspection delay of the other targets. Despite the fact that mission completion time is the specific objective of TCTPA, Figure 2.12 shows that for the considered scenario, TCTPA always completes the mission with almost twice the number of rounds used by AC-GaP.

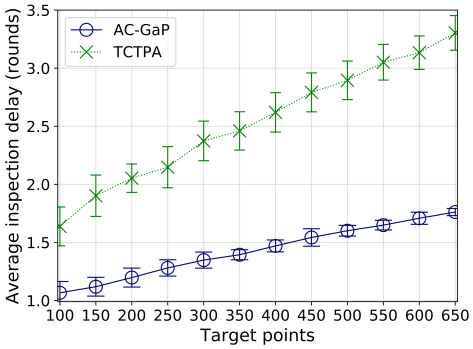


Figure 2.10. Average inspection delay in rounds.

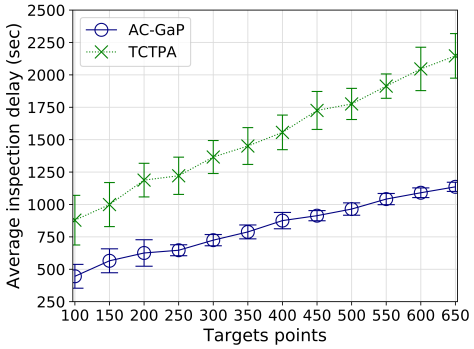


Figure 2.11. Average inspection delay in seconds.

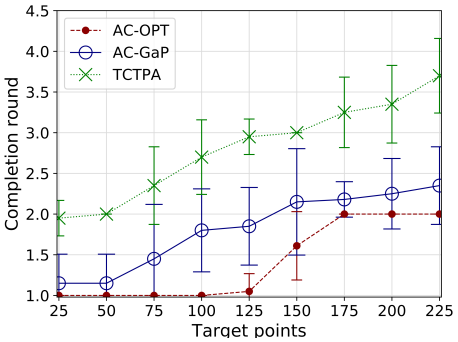


Figure 2.12. Completion round.

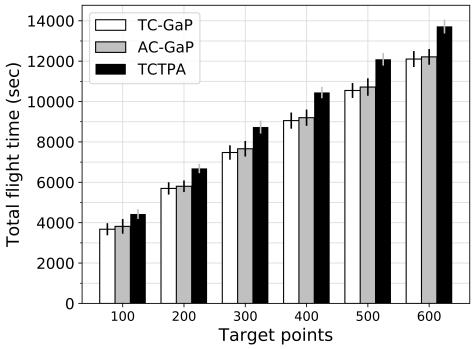


Figure 2.13. Flight time until mission completion.

Energy cost

Figure 2.13 shows the total flight time over all the 20 rounds, considering all 5 drones, for the three heuristics TC-GaP, AC-GaP and TCTPA. Notice that since we use the linear energy model discussed in Section 2.1.5, this measure directly reflects the energy expenditure of drones.

The graph shows that the difference between the algorithms in terms of total energy expenditure is negligible, although in small favor of the greedy approaches.

This explains that the superiority of the greedy approaches resides on the way drone visits are scheduled along the rounds and within single rounds, rather than in the minimization of the traversed paths.

Computation time

In Figure 2.14 we compare the algorithms in terms of computation time. For this and for all the previous experiments we adopted a homemade simulator, programmed in Python, and we ran the ILP optimization problems using the Gurobi optimizer [30]. We ran all the experiments on a Lenovo X3550 M5, with 2 CPUs Intel(R) XEON(R) E5-2650 @ 2.20GHz with 16 cores and 32 GB RAM [68].

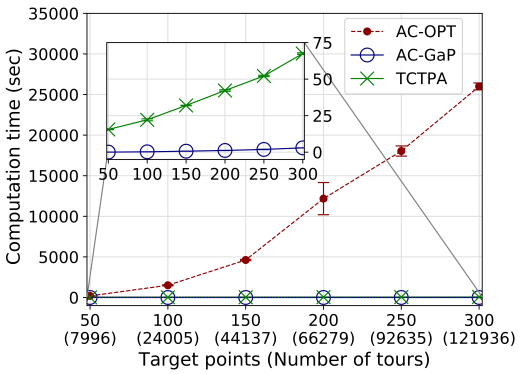


Figure 2.14. Algorithm computation time.

In order to stress the optimal algorithm while keeping the problem size small enough to be executed in a reasonable time to have enough runs for a reliable experimental evaluation, we designed a new setting with 10 drones, and a growing number of target points, from 50 to 300. This experiment shows that the computation time of the optimal solution grows prohibitively high also for this small a setting. With 300 target points the computation time is already in the order of 7 hours. Such a high processing time makes the optimal solution inapplicable to emergency critical applications which require prompt intervention of the aerial network and do not tolerate delays. The figure also contains a zoomed plot which shows the difference between TCTPA and AC-GaP in terms of running time. Though negligible, this performance parameter is also in favor of AC-GaP.

2.1.6 Real field experiments

In this section we experimentally investigate the applicability of our proposal and validate simulation results. The equipment we use for the test-bed includes a fleet of drones — able to follow way-point missions using GPS coordinates — and, a laptop — needed to run the algorithms, compute the trajectories and to provide them to drones.

Figure 2.15. DJI F550 Hex-rotor with Naza-M V2

First, we discuss the major challenges we dealt with while performing the mission. Then, we compare the simulation results with real-field experiments, under the same settings. Finally, we compare our algorithms against state-of-art solutions. We evaluate two scenarios, with battery replacement and with battery recharging between consecutive rounds. To conclude the analysis, we also use our approach to address area coverage, i.e., a different objective function for which the algorithm is not specifically designed.

Real-field Scenario

In the experiments we simulate a rescue mission with several survivors needing help. The experiments are performed in a university soccer field in Amman, Jordan, with an area of interest of around $140m \times 90m$, with $|\Psi| = 40$ randomly distributed

target points, modeling survivors. For each experiment, we perform 3 different runs. Where not otherwise stated, we use a fleet of four drones, with four different depots, located at the borders of the field, 5 meters apart from each other. The drones are DJI Flame Wheel 550 (F550) [55] hex-rotor combined with a Naza-M V2 [69] flight control system, as showed in Figure 2.15. The Naza-M V2, through the GPS module, allows drone localization and enable way-point missions. All the drones were equipped with LIPO batteries, 3500 mAh, 11.1 v, 25 C which allow around 7 minutes of consecutive flight with a speed under 5m/s. The time required for a full recharge is around 1 hours. Notice that, temperature can affect the LIPO battery's performance. Most of the experiments were carried out in February, with a temperature from 7°C to 15°C.

We consider a maximum number of rounds $N = 7$. After the end of each round, we recharge or replace the drone batteries and measure the required times. These times are then used to evaluate two scenarios in which drone batteries are: a) replaced with fully-charged batteries, or b) recharged, between consecutive rounds.

Table 2.2 lists all the relevant experimental parameters.

Table 2.2. Experiment settings

Field	Jordan University Stadium, Amman, Jordan
Local time	9.00-17.00 a.m.
Local temperature	$+7 - 15$°C
Wind Speed	1.0 to 2.1 m/s
Field Size	140 x 90 (meters)
Number of drones	4
Number of targets	40
Hovering time	90 seconds
Height above the ground	4 to 10 meters
Max number of rounds	7

Experiment workflow. To reproduce a real application scenario, we take the following steps:

- We perform a preliminary computation of the set of candidate trajectories for each drone. We compute these trajectories by using the Algorithm 2, which is also adopted in the simulations.
- We run the algorithms AC-OPT, AC-GaP and TCTPA to obtain the selected trajectories and their schedule in rounds.
- We let the drones fly along the selected trajectories, taking measures at each depot and target point.

Moreover, as experiments are conducted on a flat soccer field, we consider flight trajectories such that drones vertically take off, fly each at a different constant height, and vertically land to the depot.

To precompute candidate trajectories we assume that drones fly at constant speed when moving from target to target, and hover for about 90s above each

inspected target. To take into account the different speed that the drones may reach during flights of different lengths, we consider a uniform speed of 1.3 m/s for flight distances of up to 30 meters, 2m/s up to 50 meters, and 2.5m/s for longer distances.

We model the drone energy availability on the basis of the information advertised by the producer and our previous experience: around 450 seconds of flight-time when the speed is up to 5m/s. We adopt this value by considering that in similar experiments the drones never exceeded 5m/s of speed. It is worth to notice that these values affect the set of feasible trajectories in any algorithm that takes energy limitation into account. Nevertheless, the errors in this preliminary phase have a negligible impact in the algorithm comparative evaluation, with inspection delays that are just few seconds away from the offline calculated estimate.

Mission issues and challenges

We now discuss the major challenges we faced while deploying drones and performing missions.

Aerial collisions. To avoid potential collisions between drones, we used different flight heights for each drone. We underline that thanks to the pruning algorithm phase (2.1.4), two drones never visit the same target, which reduces collision risks considerably. In the experiments for which multiple drones shared the same depot, take off and landing procedures must be performed in a serial schedule. Moreover, drones can collide with physical objects. In this case two solutions are possible. First, drones may have integrated obstacle avoidance systems [54]. Second, drones can integrate an online collision avoidance mechanisms in the UAV control pipeline, to dynamically change the trajectories. For example, the work in [70], presents an online avoidance maneuver that alters the reference trajectory of the vehicle by adding an offset in height in case of a possible collision.

In the proposed presented in this section, we did not have any physical obstacle along the field, and each drone had its own depot.

Errors in the candidate trajectory pre-computation. As drones have limited energy, estimating their flight capabilities is a significant challenge for a successful mission. If the algorithms underestimate the energy consumption of drones, they may produce unfeasible paths, leading to emergency landings. As drones manufacturers give little information on the energy consumption model — they typically describe the device energy availability in terms of indoor hovering time or flight at uniform speed — we measured the drones' capabilities by analyzing real field past experiments. Specifically, when computing the set of candidate trajectories χ_u for each drone u, we estimate the flight capabilities by averaging and interpolating measures obtained in similar environmental conditions, considering the different activities such as take-off, flight at uniform speed, hovering and landing. Then, we slightly reduced the value of this estimate to have a conservative evaluation of the drone capabilities to make the solution more tolerant to unpredictable conditions, such as adverse wind.

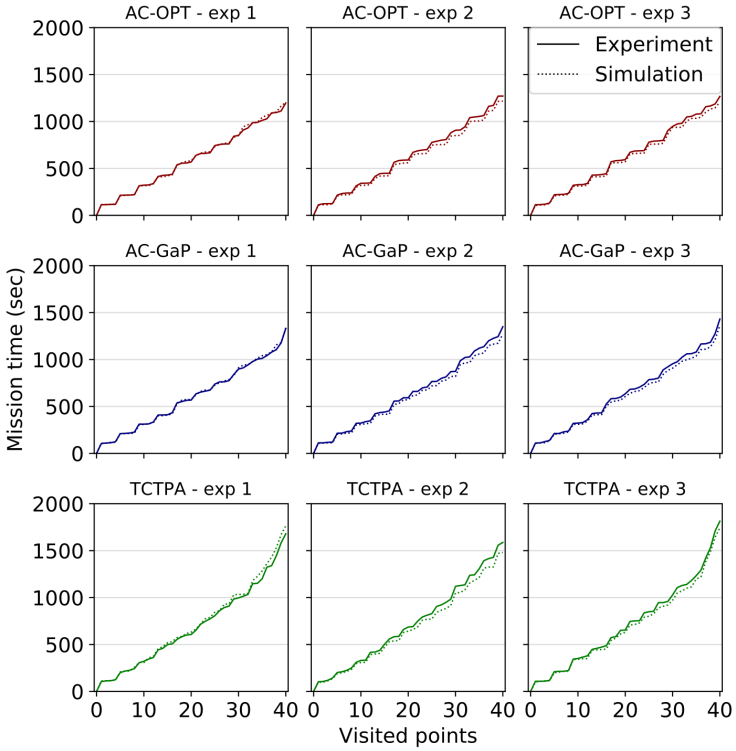


Figure 2.16. Simulation vs Experiment visit times.

Validation of simulation results

We now compare simulations and experiments conducted on the real test-bed using the same settings described in Table 2.2. We do not show a comparison between simulated and real experiments in terms of rounds as they produce the same results. Figure 2.16 shows the time elapsed (y-axis) until the visit of an increasing number of target points (x-axis) for all the compared algorithms. For this experiment, we considered zero waiting time between two consecutive rounds. The dotted lines represent the offline simulated trajectories, while the solid ones are the real field experiments. The lines show only very small differences between simulations and experiments. Confirming this analysis, Figure 2.17 evaluates the difference between simulations and experiments in terms of the Mean Absolute Percentage Error (MAPE). The figure shows that the MAPE of the average inspection delay is always below 7%. This slight discrepancy is mostly due to uncontrollable environmental factors (e.g., wind), measurement errors, and drone acceleration/speed in the real field.

Thus, our first set of experiments validates simulation results.

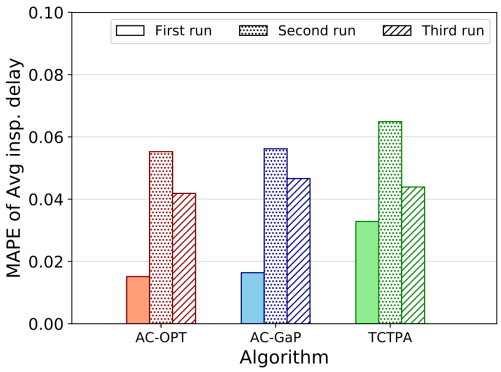


Figure 2.17. Simulation vs Experiment average inspection delay in seconds.

Progressive coverage evaluation

We experimentally compare our algorithms with previous solutions. We evaluate two scenarios, which differ in the operation between two consecutive rounds: the first implements battery replacement while the second provides battery recharging. Obviously, battery recharging has a significant impact on overall mission time.

Figure 2.18 shows the progressive coverage percentage for the three algorithms (see section 2.1.5), with respect to mission rounds. Specifically, AC-GaP performs very well, close to AC-OPT. Both algorithms achieve very high coverage in the early rounds of execution, with complete coverage before round 3 for AC-OPT and round 4 for AC-GaP. Instead, TCTPA performs poorly, as it requires 7 rounds to complete the visit.

We further investigate progressive coverage in the two scenarios in terms of elapsed time from the mission start.

In the battery replacement scenario, shown in Figure 2.19, AC-OPT and AC-GaP perform better than TCTPA, reducing the average inspection delay of about 25% (see Table 2.3). This gap heavily increases in proportion to the time spent at the depot for battery replacement and device maintenance, due to the higher number of rounds required by TCTPA.

If we consider battery recharging between rounds, as we show in Figure 2.20, the performance of TCTPA degrades significantly with respect to AC-OPT and AC-GaP, which however present a step-wise increase in progressive coverage percentage. TCTPA takes 5000s to cover about 50% of targets, while at the same time, AC-OPT and AC-GaP cover 75% and 72% of targets, respectively.

Table 2.3. Average Inspection Delay on real-field experiments

Average Inspection Delay	**AC-OPT**	**AC-GaP**	**TCTPA**
# of rounds	0.85 ± 0.02	0.9 ± 0.04	2.32 ± 0.13
# of seconds - with battery replacement	727 ± 15	744 ± 21	992 ± 11
# of seconds - with battery recharging	3578 ± 62	3717 ± 127	6147 ± 247

Finally, Table 2.3 shows the average inspection delay (see section 2.1.5) with respect to the number of rounds and time elapsed from the mission start, in both the battery replacement and recharging scenarios. The results clearly show that

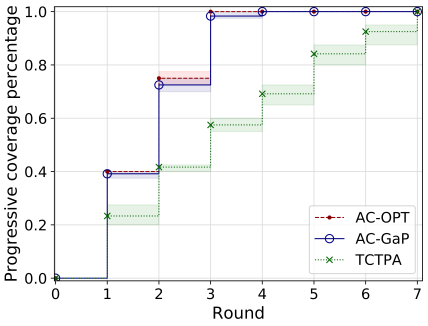


Figure 2.18. Progressive Coverage (Rounds)

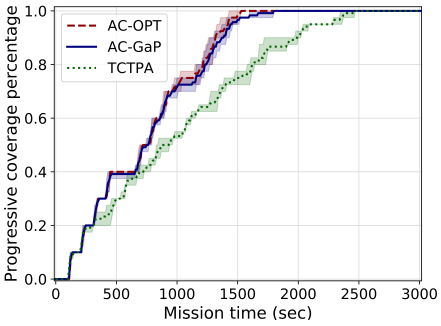


Figure 2.19. Progressive Coverage with battery replacement (sec)

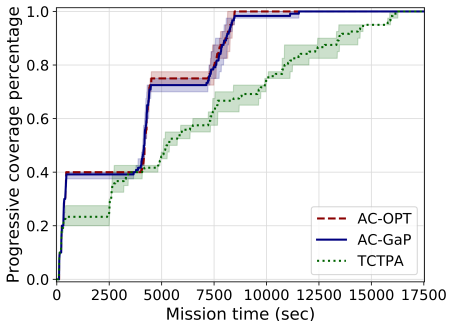


Figure 2.20. Progressive Coverage with battery recharging (sec)

AC-OPT and AC-GaP are comparable and outperform the TCTPA. In detail, the optimal and greedy algorithms perform better than TCTPA, reducing the average inspection delay in seconds of around 25% and 40%, with battery replacement and recharging, respectively.

To conclude, our second set of experimental results confirm the real field applicability of our approximation algorithm, which outperforms TCTPA. The experiments also reveal that the new WPC metric correctly reflects the algorithm capabilities to provide early target inspection, especially in battery recharging settings.

Energy model - sensitivity analysis

As we previously discussed when we addressed the challenges of a real field test-bed, errors in the energy model may result in the computation of inefficient or infeasible trajectories due to an incorrect estimate of the drone capabilities. We studied the effects of under-estimating and over-estimating the drone energy consumption considering different error values. Due to the randomness of the target deployment over the area, in none of the experiments we had a situation in which drones had the exact energy availability to complete their trajectories with no residual energy. Therefore the algorithms showed a good flexibility to errors in the energy model. We had to introduce an error as high as the 30% before being able to observe significant differences in number of rounds with respect to the case of a correct energy model.

In a scenario with 20 randomly deployed target points and 4 drones an overestimation error of 30% in the drone energy consumption produces shorter trajectories than necessary, requiring 1 additional round in all the algorithms. In contrast, an underestimation error of 30% produced unfeasible trajectories, so that some drones were not able to complete their tasks and had to return to the depot, requiring additional time to identify the un-visited targets and re-compute a solution to complete the mission.

Area coverage evaluation

In this section, we investigate the performance of our approach when the application requires coverage of an entire area, and not just of some target points. To extend our target coverage algorithms to the new application scenario, we provide a square tessellation of the area (with tile sized such that they can be inscribed in the drone monitoring range) and set a target at the center of each tile with null hovering time requirement. With this setting, by inspecting each target, a drone entirely covers the area of interest. For this evaluation, we consider an algorithm specifically designed to obtain area coverage: the cooperative *Sweep* algorithm — an extension of the approach proposed in [53] to drone networks. The algorithm partitions the whole area among the available drones, taking into account their capabilities and different initial locations. Each partition is then covered with a rectangular scale-line pattern.

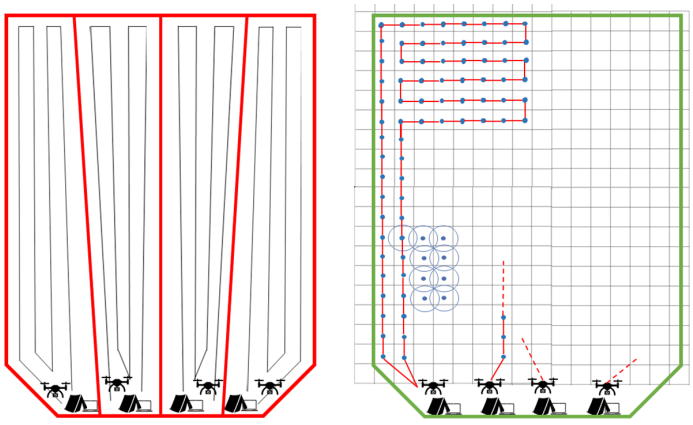
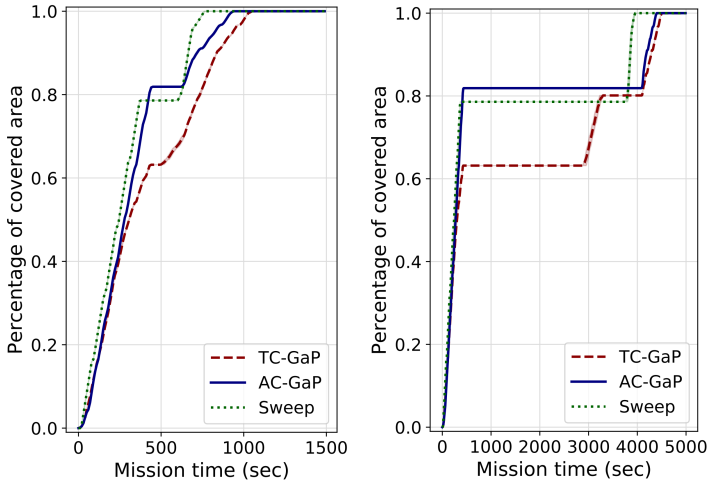


Figure 2.21. Cooperative coverage path planning sample scenario.

The drawing of Figure 2.21 shows an example of the algorithm trajectories in the addressed scenario. The left image represents the *Sweep* algorithm. Notice that, if a path requires to much energy to be executed by a drone in a single trip, we truncate it and we reassign the remaining part to a subsequent round. The right image exemplifies the setting of target points to address this scenario with target coverage algorithms.

We run two different experiments to test the algorithm performance while covering the whole area with 4 drones, both with replacement and recharging actions between consecutive rounds. We consider a 360°-camera with a monitor range of 2.5 meters. We do not consider hovering time, but we slow down the maximum speed to 2m/s, to allow the camera to collect data during the flights.



(a) Battery replacement (b) Battery recharge

Figure 2.22. Percentage of covered area.

In this scenario we compare Sweep with our greedy algorithms: AC-Gap, which employs the accumulative coverage metric, and TC-GaP, which employs the total coverage metric. Figure 2.22(a) and 2.22(b) show the percentage of covered area since the beginning of the mission, in case of battery replacement and battery recharge, respectively. While all the algorithms have the same performance in number of rounds to achieve total coverage (i.e., they cover the whole area in 2 rounds), AC-Gap covers around 82% in the first round, while Sweep only the 78% and TC-GaP the 62%. In particular, the figures show that AC-GaP visits most of the points at the beginning of the mission, which is extremely helpful in critical situations, especially when the batteries are being recharged. In Figure 2.22(b) AC-GaP visits the 82% of the points in the first 8 minutes of mission, while Sweep needs more than 1hour to reach the same coverage percentage.
Finally, we note that despite the performance differences shown so far, the three algorithms need about the same time to achieve a complete coverage, for both the battery replacement and battery recharging scenarios. Sweep needs 800s and 4000s, AC-GaP needs 900s and 4350s, and TC-GaP completes the mission in 1050s and 4400s, respectively, in the two scenarios.

2.1.7 Conclusions

The work addresses the problem of assigning location based tasks to a fleet of drones in an emergency critical scenario, to ensure early inspection of target locations. We propose a novel metric, called weighted progressive coverage, to measure the capability of a trajectory planning algorithm to provide early target inspection, and formulate a related optimization problem. We show that weighted progressive coverage generalizes traditional metrics of coverage, as well as a new notion of accumulative coverage which is directly related to early coverage capabilities. Due to the NP-hardness and high computation time of the optimal approach, we propose

a new polynomial time algorithm with a constant factor approximation of 1/2 of the optimal. We study the proposed algorithm by means of extensive simulations, showing that it outperforms previous approaches in terms of all the metrics of coverage, average inspection delay, energy consumption and computation time. We also validate our simulation results through prototype experiments and demonstrate the applicability of our approach in real field scenarios.

2.2 GenPath - A Genetic Multi-Round Path Planning Algorithm for Aerial Vehicles

In the previous section we highlighted the importance of dealing with limited energy and capability of drones. We monitored target points in multiple consecutive trips, allowing battery replacement and data offloading in between.

In this work we further analyze multi-trip missions into a wider range of applications. We study the use of a Genetic Algorithm to adapt and schedule trajectories into a multi-trip mission, dealing with applications and drones' constraints. In fact, as already mentioned, there have been several proposals for task assignment which consider a one-trip per drone assignment, with the purpose of maximizing the number of inspected targets, or minimizing the task completion time. But drones have limited and heterogeneous energy and storage availability, and may not be able to complete the assigned path with a single trip. Existing approaches stop at this point — they do not tackle the problem of dealing with multi-round missions to provide complete monitoring coverage, with battery replacement/charging and data offloading in between. A drone may start from a depot, reach some points of interest, go back to the depot to recharge or replace the battery, and then continue the mission, exploring other points of interest.

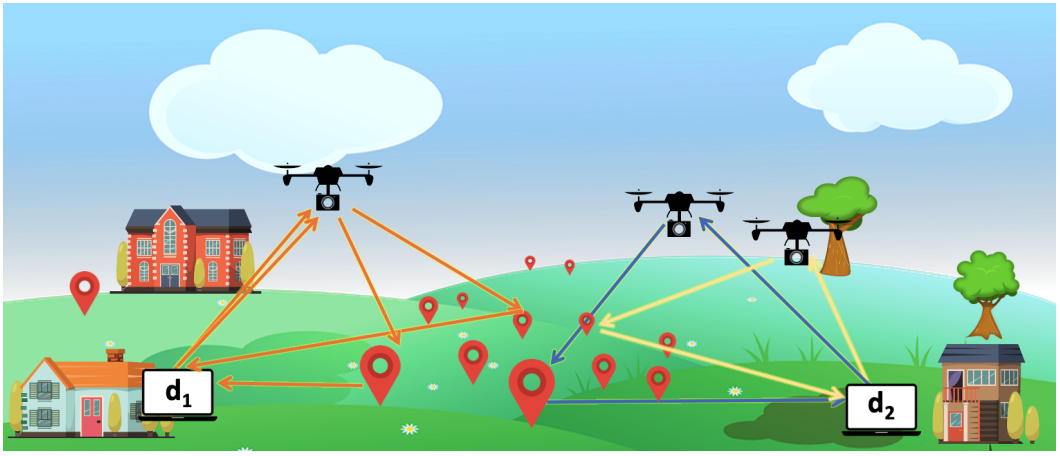


Figure 2.23. Example of a mission: two different depots (d1 and d2), a set of three heterogeneous drones, and several critical points to visit (i.e., the red markers) and multiple tours for the same drone (orange arrows).

Although most of the available path planning algorithms may be adapted to deal with multi-round missions (e.g., they can split a mission in multiple tours allowing battery replacement), path assignment is not immediate and should be optimized. In particular, the paths should be assigned in an optimized way, according to the objective function. For example, for early coverage objective, the tours should be optimized such that the majority of points are covered in the earliest tours. While for total coverage objective, tours should be optimized so that most of points are visited within the end of mission.

To this end, we propose Gen-Path, a genetic algorithm that optimizes path scheduling in multi-round missions. Gen-Path takes as input a set of feasible paths

for each drone, generated by other path planning systems. Then, through selection, crossover, and mutation, Gen-Path optimizes the solutions, finding a near optimal assignment of tours under any objective function. Differently from other proposals, Gen-Path is able to optimize already defined paths and deal with limited energy constraints.
We implemented Gen-Path on DEAP [71], an evolutionary computation framework developed in python. We analysed its performance and compared it to related algorithms. Simulation results demonstrate that Gen-Path can fit different scenarios, characterized by different objective functions, decreasing the energy consumption, with respect to related solutions, up to 30% with 12 drones, while inspecting comparable number of targets. A distinguishing aspect of Gen-Path is that it can easily incorporate paths that present restrictions, such as paths imposed by public authorities: in some areas no fly-zones may define the only viable paths.
To summarize, the key contribution of this work is a genetic based path selection algorithm, called Gen-Path, which integrates and adapts single-path missions to multi-round missions, eventually improving their performance, while dealing with real scenario constraints.

2.2.1 Problem Formulation

We consider a multi-round mission where a swarm of drones, located at different depots, will perform monitoring operations to inspect an area of interest with several target points (see Fig. 2.23). Drones can offload data and replace/recharge batteries between consecutive rounds.

Let N be an upper-bound to the number of rounds. Notice that, N may reflect the hours available to complete the mission, or the number of recharging phases or the batteries available for the drones (i.e., the maximum possible consecutive rounds).

We denote with $\mathcal{U}$ the set of drones and with χ the input set of possible candidate tours. We recall that this formulation allows us to adapt most of the literature path planning algorithms to multi-round missions. Our proposal allows to assign trajectories to drones in the rounds according to a given objective function. In fact, the algorithm outputs $\mathcal{S}$, i.e., an assignment of tours, maximizing a given score function $f(\mathcal{S}) \rightarrow R$. This general definition allows the proposed genetic framework, Gen-Path, to be adapted upon need, and in several different scenarios. For example, the algorithm can consider target coverage, round based accumulative coverage, visit latency minimization, and so on.

Path assignment optimization

We now formalize the problem of path selection in terms of integer programming. We define with χ_u the set of all paths that each drone $u \in \mathcal{U}$ can traverse. χ_u is composed of only valid tours, namely tours that include the depot, namely d_u, of drone u and that can be traversed within the constraint on energy consumption imposed by the drone's battery, i.e., b_u. We consider trajectory-based variables as

follows: $z_p^u(n) \in \{0,1\}$, to denote the decision to assign path $p \in \chi_u$ to drone $u \in \mathcal{U}$ at round $n \in [0,..,N]$.

We define Problem 2.4 as follows:

$$\begin{array}{ll} \max f(\bigcup_{u\in\mathcal{U},n\in[0,N],p\in\chi_u:z_p^u=1} p) & (a) \\ s.t. & \\ \sum_{p\in\chi_u} z_p^u(n) \leq 1,\ \forall u \in \mathcal{U},\ \forall n \in [0,N] & (b) \\ z_p^u(n), \in \{0,1\},\ \forall u \in \mathcal{U}, p \in \chi_u,\ \forall n \in [0,N] & (c) \end{array}$$

Problem 2.4. Multi-Path Multi-Round Optimization.

Where constraint (b) imposes that each drone performs at most one tour at any given round n, while constraint (c) defines the binary domain of variable $z_p^u(n)$.

Notice that, the battery constraint is incorporated in the definition of the feasible trajectories: the input considers only the tours that drones can perform within their battery limitation b_u.

Problem complexity

The defined optimization problem works under any objective function $f(\mathcal{S}) \to R$, defined by the end-user. Therefore, the problem may be NP-Hard for some objective functions. Moreover, if the function is non linear, the problem may be even more complex to solve.
This hardness motivated us to seek for more efficient sub-optimal solutions, with near optimal performance. In a critical scenario the optimal solution may be impracticable, for example in earthquake emergency, wildfire, and general rescue operations where a timely intervention is extremely important.

2.2.2 Gen-Path: A Genetic Framework for Path-Planning

A brief primer of Genetic Algorithms

Genetic algorithm (GAs) is an optimization technique, firstly proposed by Holland in [72], which finds its roots in the biological evolutionary process. The key idea is that through generations only the most suitable individuals, for humans the stronger and healthier, survive, making the next generation improved with respect to the previous one. Genetic Algorithms are frequently used to find optimal or sub-optimal solutions to hard, or NP-hard, problems. A GA starts with a problem and a set of possible solutions, the population. These solutions are recombined and mutated (like in natural genetics). In this way the GA produces new child solutions, repeating for multiple generations. To each child solution is assigned a fitness value, depending on the problem's objective function. In each generation only the solutions with the higher fitness survive. The GA keeps evolving the better solutions over generations, until it reaches a stopping criterion. Genetic algorithms have become popular due to their good performance with respect to other heuristic approaches. Thanks to their evolution based model, GAs are able to exploit historical information, and to move only through promising solutions.

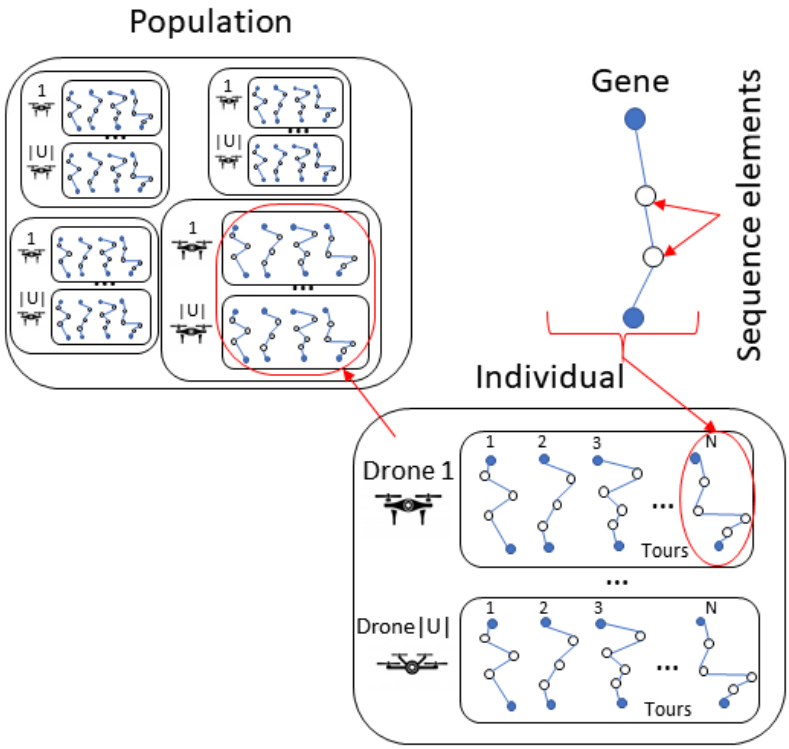


Figure 2.24. Genetic Algorithms components.

One of the main benefits of GAs for our context is the capability to optimize both continuous and discrete functions, even with multiple objective functions.

Gen-Path overview

We now describe Gen-Path, our genetic algorithm to solve the multi-round trajectory problem. The algorithm starts from a random population: a set of random assignments $\mathcal{S}_i$, composed from the input set of feasible trajectories. Each solution $\mathcal{S}$ is evolved to maximize the value of the given fitness function $f(\mathbf{S}) \rightarrow \mathbf{R}$, to finally return a near optimal solution.
This approach has two main advantages: 1) the algorithm finds near optimal solutions in a very short time; 2) through a special mutation and cross-over phase, the algorithm can generate new trajectories that can be used in the solution. In particular, this property allows to outperform other path selection approaches, as the algorithm creates also new optimized trajectories.

Notice that, the creation of new paths is strictly connected with the presence of a validation operator (see Figure 2.26) which validates each new trajectory feasibility. We will discuss better this operator in the next section.

Gen-Path components

We now define the main components of the Gen-Path algorithm. As we can see in Figure 2.24, the **individual** represents a possible solution **S** to the multi-round path-planning problem. We recall that $\mathcal{U}$ is the set of drones for which we aim at

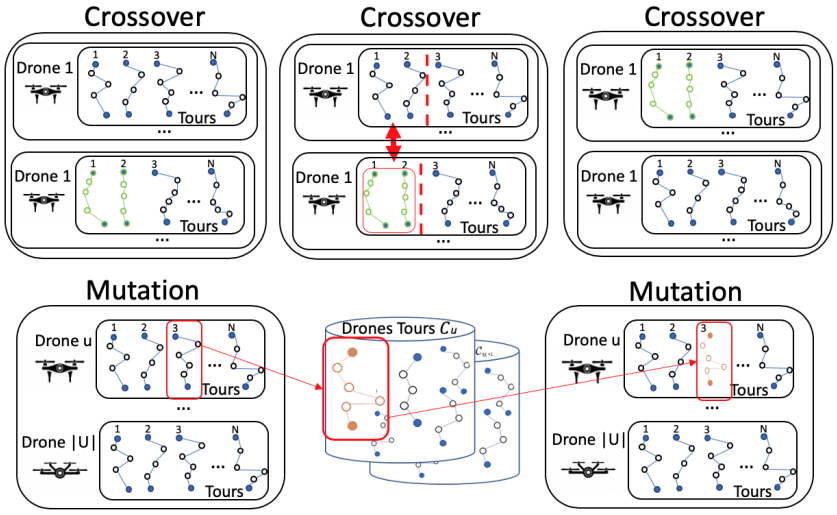


Figure 2.25. Main phase operations.

finding a set of N trajectories to visit the area/points of interest. Therefore, our individual $\mathbf{S}$ can be represented as a matrix $Nx|\mathcal{U}|$:

$$\mathbf{S} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,N} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{|\mathcal{U}|,1} & p_{|\mathcal{U}|,2} & p_{|\mathcal{U}|,3} & \cdots & p_{|\mathcal{U}|N} \end{bmatrix} \tag{2.12}$$

where each element $p_{u,n}$ represents the tour which the drone u runs at round n. Therefore, each drone will run at most N tours, with data offloading and battery management in between.

A **gene** represents a single component/feature of the solution. In our case a gene is a single tour $p_{u,n}$ of the solution (i.e., individual), namely $p_{u,n} =< d_u, t_0, t_1, ..., t_i, t_{i-1}, d_u >$, where d_u is the depot of drone u and $t_i \in \Psi$ is a target point. We denote with Ψ the set of target points to inspect. We also introduce smaller components that we call **sequence elements**. They compose the genes. In our problem settings, each *sequence element* represents a target visited by a tour (i.e., gene), namely each target $t_i \in p_{u,n}$ is a sequence element. Notice that, the sequence elements are introduced to increase the solution performance. They allow to change also the tours that are given as input (i.e., genes), opening the solution to completely new opportunities.

Finally, the **population P** is a set of individuals, which iteratively evolves driven by a biological evolution process. We define as $\mathbf{P}(0) = \{\mathbf{S}_0, \mathbf{S}_1, ..., \mathbf{S}_k\}$ the initial population of k-elements, which is randomly generated, allowing different range of possible solutions. During each successive generation t, we define with $\mathbf{P}(t)$ the current population.
The fitness function $f(\mathbf{S}) \rightarrow \mathbf{R}$ evaluates the solution domain.

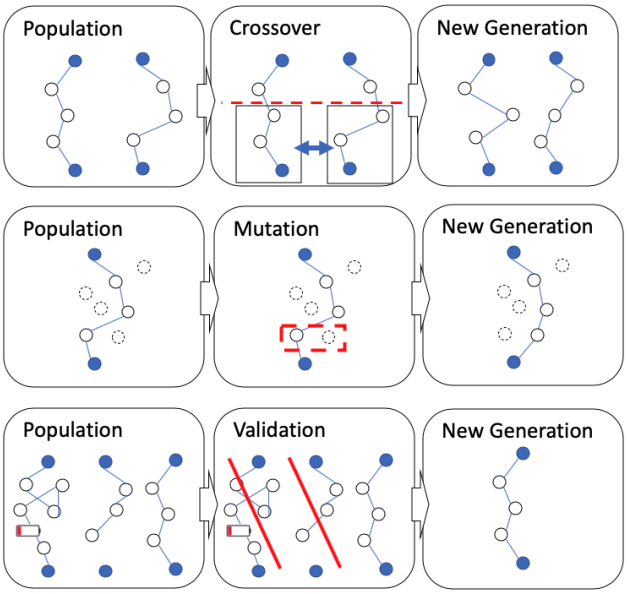


Figure 2.26. Secondary phase operations.

Genetic Algorithm operations

We have two main evolutionary phases: 1) the main phase, based on individuals and genes to create new solutions; 2) the second phase, based on *gene* and *sequence elements* to increases genetic diversity and creates new tours (genes) in solutions. We recall that the solution $\mathbf{S}$ is an assignment of N-tours for each drone $u \in \mathcal{U}$.

Main phase We now define the **cross-over** and the **mutation** for individuals, shown in Figure 2.25. The **cross-over** operation exchanges tours between two different individuals (solutions). It selects a random crossover point in the list of N-tours of drone u; then, in the two solutions, the tours to the right of that point are swapped. This results in two offspring, each carrying some genetic information. The **mutation** operation randomly exchanges a tour from the solution of drone u by selecting one from its feasible trajectory χ_u. This operation helps to increases the genetic diversity and add new tours in the solution.

Secondary phase The secondary phase directly changes tour (gene) structures and target points (sequence elements). It involves three main operations, the **cross-over**, the **mutation**, and the **validation**. Figure 2.26 shows the three operators. The **cross-over** operation randomly combines tours from same drone u. Given a tour t, it selects a random crossover point in the tours and swaps them. The targets to the right of that point are swapped. This results in new tours and solutions, each carrying some genetic information. The **mutation** randomly exchanges tour targets. For a given tour $p_{u,n}$ a target t_i is randomly chosen and substituted with a random one $t_j \in \Psi, j \neq i$. As this secondary phase modifies tours, the **validation** operation allows to validate tours. It checks the battery and capability constraints (i.e., if a tour $p_{u,n}$ is feasible for drone u). Notice that, this operator can include any energy or assessment model to check the tours.

Algorithm

The Gen-Path algorithm is presented in Algorithm 3. It contains the two main evolutionary phases.

Algorithm 3: Gen-Path Algorithm

```
1   Input: The set 𝒰 of drones, the candidate cyclic trajectories of drones
    {χ_u : u ∈ 𝒰}, the maximum number of rounds N, and the fitness function
    f : S → ℝ
2   Output: an assignment of |𝒰|xN tours S.
3  P ← initialize population;
4  gen ← 0;
5  while not stopping criteria do
6  |  parents ← selection(P(gen), f);
7  |  gen += 1;
8  |  — Individual Evolution —
9  |  P(gen) ← elitist-selection(parents);
10 |  P(gen) ∪ cross-over(parents);
11 |  P(gen) ∪ mutation(parents);
12 |  — Gene Evolution —
13 |  if Gene-evolution is enabled then
14 |  |  P' ← gene-cross-over(parents);
15 |  |  for i ∈ P(gen) do  Mutation loop
16 |  |  |  P' ∪ {gene-mutation(i)} ;
17 |  P(gen) ∪ validation(P');
18 return best-individual(P(gen))
```

The algorithm starts with a random initialization of the population, which evolves through several evolutionary operations. Each operation is performed on a set of individuals according to a random probability. At each iteration, the algorithm chooses some individuals as parents to generate the next generation, based on the fitness function. Some of these are directly included in the new population without changes, to avoid loss of performance (i.e., a good solution may be lost or degraded), while others are mutated and changed. In the main evolutionary phase, these individuals are changed at the gene level: some of the parents are randomly chosen for cross-over and mutation. In the second evolutionary phase, some genes (tours) of the new individuals (solutions) are selected. Their *sequence elements* (targets) are mutated and combined to increase genetic diversity and investigates new possibilities. Therefore, new trajectories are built, and they are validated by the *validation* operator.
Notice that the validation can be a function $f(p, u) \to [0, 1]$ which takes a drone u and a new path p and validates its feasibility (e.g., if the battery constraints are meet). This function is required to enable the secondary evolution, otherwise this phase is disabled. The secondary evolution may be disabled also when the tours can not be changed (e.g., due to strict regulations which define the only viable paths).
The algorithm continues and iterates to modify the population and takes and shares only better proprieties, according to the chosen optimization function. It stops when it reaches a *stop criteria*, which may be defined upon need. The stop criterion is usually based on: the max number of generations, the convergence of the performance,

or a score threshold for the solution. Finally, the output is a sequence of N-tours for each drone, such that the overall mission maximizes the given input function.

2.2.3 Performance Evaluation

We now evaluate the performance of Gen-Path. We compare it with a Greedy based algorithm, TC-GaP (see Section 2.1), which considers a multi-round optimization with *total coverage* objective function. The *total coverage* aims at providing target inspection (i.e., maximize the number of targets covered in the mission). We calculate the input set of feasible trajectories χ as proposed in Section 10, which uses Christofides approximation algorithm [15]. Given a set of target points Ψ, for each drone u, it builds a Traveling Salesman Problem (TSP) approximated solution. Thus, it determines an ordered sequence of the target points, which splits into several parts, to exclude unfeasible ones and those that do not meet the energy battery limitations b_u of drone u.

We evaluate the following metrics: ***Covered points*** defined as the number of target points covered within the mission; ***Average Tour Cost*** defined as the average time in seconds taken by each tour (i.e., we consider that the drones energy consumption corresponds to flight time, with one energy unit consumed for each second of flight ($1e_u$)).

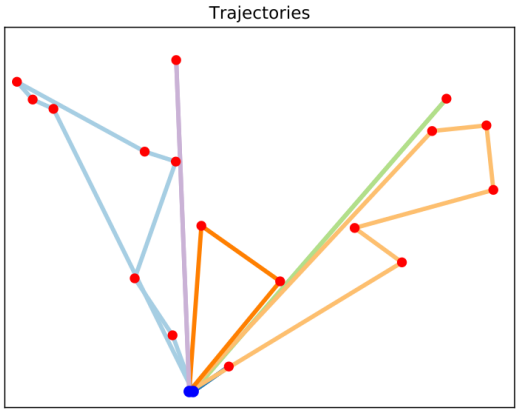


Figure 2.27. Gen-Path path example.

Scenario

We consider a scenario with $t = 200$ target points randomly distributed on an area of 5x5km. We consider 4 to 12 drones starting from different depots, with 10 minutes of autonomy, flying at an average speed of $8m/s$. We consider $N = 3$ maximum mission rounds (i.e., the consecutive flights allowed to each drone).
We consider a *stopping criteria* based on the maximum generation number, that we set to 150 generations (i.e., Gen-Path terminates after 150 evolutions and outputs the best individual found). Figure 2.27 shows an example of paths generated by Gen-Path in an area of 2x2km, with 2 drones. Red marks represent points of interest (POIs) that must be explored, while the blue mark represents the depot where UAVs depart, recharge their batteries and land. Each drone starts the mission by exploring

some POIs; it returns to the depot for recharging its battery and off-loading data; eventually it continues the exploration of other POIs.

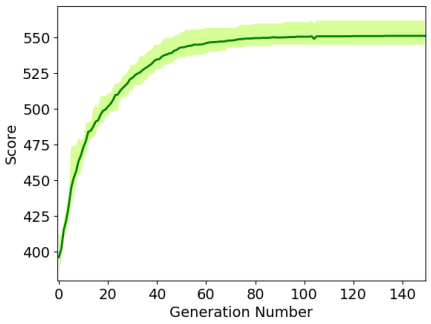


Figure 2.28. Gen-Path convergence.

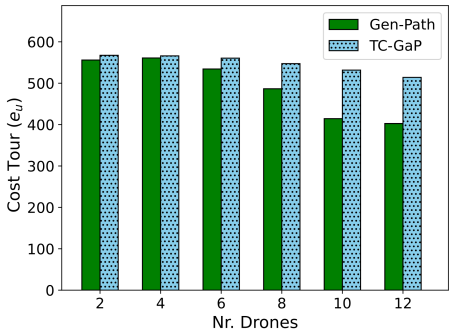


Figure 2.29. Average Tour Cost.

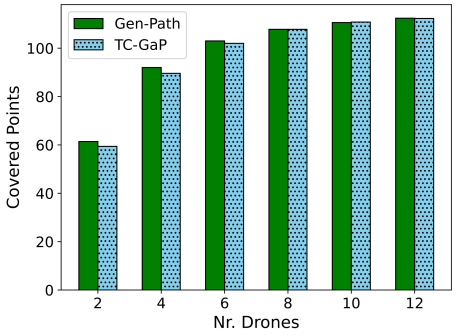


Figure 2.30. Number of covered target points.

Performance

Figure 2.28 shows the convergence of Gen-Path for the *total coverage* objective function, in case of 4 drones. Gen-Path converges to a optimal solution with less than 80 generations.

Figure 2.29 shows the performance of Gen-Path under the objective of minimizing the length of the tours while maximising the covered points. The figure shows how Gen-Path notably reduces the length of the tours, resulting in a lower energy consumption for the drones. By increasing the number of drones Gen-Path reduces the tour cost from 5% in the case of 2 drones up to 30% in the case of 12 drones, keeping the number of covered points comparable. In particular, Figure 2.30 shows the number of covered target points by Gen-Path respect to TC-GaP. It shows that Gen-Path is always comparable, while it also slightly increases the performance for small scenarios.

The motivation behind such improvement is given by the ability of Gen-Path to optimize more complex objective functions. Notice that, while TC-GAP has been specifically developed to optimize coverage, our approach can use any given objective function. Therefore, these results confirm the Gen-Path applicability, and, thanks to its *Gene-Evolution* approach also its superiority in several scenarios. In fact,

Gen-Path is able to create new paths, increasing the number of trajectories and the points that can be visited.

2.2.4 Conclusions

We develop Gen-Path, a genetic algorithm for efficient scheduling of multi-round UAV missions considering the limited capabilities of drones. In fact, most of the existing works do not consider the limited autonomy of drones, precluding their applicability in real scenarios.
Gen-Path leverages the existing plethora of algorithms for path planning to build multiple tour candidates and schedule them into a multi-trip multi-drone mission. In particular, it enables optimized multiple consecutive trajectories for each drone, with data off-loading and battery recharging in between.
We evaluate Gen-Path against a state of art algorithm, namely TC-GaP. Gen-Path achieves the same coverage in less time, improving the performance up to 30% in terms of energy cost.

Chapter 3

Connected Trajectory Planning

In the previous Chapter 2 we analyzed the problem of path planning for fleets of multiple drones without considering the communication capabilities of the drones. Conversely, in this chapter we propose two novel solutions that explicitly incorporate communication in the design of path planning algorithms, to address a wider range of scenarios. We focus on safety critical missions, including post-disaster areas, or military fields, requiring prompt area monitoring and fast detection of events of interest. Moreover, we do not rely on any communication infrastructures (e.g., 5G), which can be (a) unavailable or disrupted in harsh environments [22], (b) inadequate to meet the data rate and delay requirements of a safety critical application. Instead, we consider networks of drones that communicate and are connected in ad-hoc manner, namely Flying Ad-hoc Networks (FANETs).

In Section 3.1 we formulate the *connected deployment problem* that requires to urgently monitor a set of target points while streaming data to a base-station. We leverage communication capabilities of drones and we require the FANET to create connected formations to ensure multi-hop low-latency communication while performing the monitoring task. We show that addressing the above problem to maximize event coverage is NP-hard and we propose a polynomial time solution, called Greedy Connected Deployment (GCD), based on a two phase approximation of the problem. By means of extensive simulations and real field experiments, we show that our approach outperforms existing solutions to the related problem, both in terms of monitoring accuracy and system responsiveness.

In section 3.2 we relax the assumption of perfect knowledge of ongoing events and their location. We consider a safety critical application in which the events' time and position can only be estimated with some *uncertainty* (e.g., survivors after an earthquake). We exploit drones' communication capabilities to share local observations and build a dynamic probabilistic map of ongoing events. We integrate such a map into a virtual force approach for a joint solution to distributed dynamic trajectory planning and collision avoidance. Through extensive simulations and real-field experiments, we show that our proposal discovers new events 30-40% faster than the other algorithms, and outperforms them in terms of percentage of visited events and inspection delay, under a wide variety of scenarios.

This chapter has been extracted from the works in [21] and [23].

3.1 On connected deployment of delay-critical FANETs

The use of Flying Ad-hoc NETworks (FANETs) in emergency critical scenarios offers a powerful tool for ensuring inspection of hostile or hazardous areas, where human intervention is unsafe if not utterly impossible. In these settings, the existing cellular infrastructure may be completely knocked down, and drones may receive monitoring tasks and related trajectory plans from a master base station through long range communication means. However, long range communications typically allow limited data rate. In order to transmit the rich information necessary to perform proper assessment of the ongoing events, the drone squad may use high data-rate communications, allowing multi-hop communications with the base station. In this work, we tackle the *connected deployment problem*, namely we study how to deploy a team of drones so that their trajectories dynamically create connected formations through which the FANET can inspect a set of targets of interest, while ensuring low latency, high data-rate multi-hop communications with the master base station, under the delay-sensitive requirements of safety critical scenarios.

Previous solutions to the trajectory planning problem for teams of drones include: graph algorithms [17, 37], virtual force based approaches [73], optimization problems [18, 19, 44], bio-inspired, genetic and machine learning algorithms [74, 75]. None of these works tackle the communication problem. They are all based on the assumption that applications are delay-tolerant, and that information related to the inspected targets can be delivered when drones return to their base station or fly in its proximity.

The problem of path planning under connectivity requirements was studied for multi-robot exploration scenarios, where several terrestrial robots aim at exploring an area of interest, or at visiting a set of target points, while connected to the base station [76–79]. However, these works are based on assumptions on device mobility and energy consumption that are not applicable to the case of aerial devices. Other works [80–82] aim at preserving a swarm connectivity to the base station by controlling the device formation according to the theory of formation rigidity. Few works explicitly focus on designing drone trajectories while preserving ad-hoc, high data-rate communications among multiple drones, either considering continuous, periodic, or best-effort connectivity [83–87].

Unlike previous works, we jointly address the monitoring and communication needs of emergency critical applications, while considering the specific operational setting of a drone team. We show that this problem is NP-hard, and propose a polynomial time solution, called *Greedy Connected Deployment (GCD)*, based on two phases, the first designing target formations to maximize target coverage, and the second scheduling the formations for minimal completion time.

We compare the performance of our approach to the optimal solution and to two previous approaches. The first approach, called *AC-GaP* (see Section 2.1), is a recent algorithm for target inspection with multiple drones that aims at optimizing the cumulative coverage, but does so without constraining connectivity during the inspection. The second approach, called *2S-APX* [77], is designed for multiple terrestrial robots, and provides recurrent connectivity but neglects energy consumption, which is a critical aspect for aerial devices.

Extensive simulations show that GCD performs close to the optimal solution, but with negligible computation time. The experiments also highlight the superiority of GCD to AC-GaP and 2S-APX in terms of achieved coverage, communication latency, and computation time. In all the simulations GCD provides immediate communication of the data related to the inspected targets, while inspecting about 20% more targets, with negligible movement overhead to realize connected formations.

Real-field experiments confirm that our approach provides higher inspection accuracy and system responsiveness than AC-GaP and 2S-APX.
The novel contributions of our work are the following:

- We introduce the *connected deployment problem* which jointly addresses trajectory planning and low-latency multi-hop communications for FANETs in emergency critical scenarios, and prove its NP-hardness.
- We give an ILP formulation of the above problem, with fine-grain trajectory design.
- We provide a polynomial-time algorithm, called Greedy Connected Deployment (GCD). We also adapt two existing approaches, AC-GaP and 2S-APX, to realize emergency-critical FANET deployments.
- We perform extensive simulations showing that GCD performs close to the optimal, and better than AC-GaP and 2S-APX in terms of coverage accuracy and communication latency, as well as computation time.
- We confirm the superiority of our approach in the considered scenarios, through real-field experiments.

3.1.1 Problem formulation

In this work we tackle the problem of defining trajectories for a squad of drones so that they can monitor targets while being in multi-hop communication with their base station, to ensure prompt anomaly detection and related communication to the base station.

Let U be the set of homogeneous drones composing the squad, with a common base station location σ. Each drone is allowed to fly within the limits of energy availability of its battery b. The drone consumes energy for movements, hovering, monitoring, and for communications (both for sending and receiving) according to a device-specific energy model.

We consider an Area of Interest (AoI) where the drones of U are required to monitor a set $\Psi_{\mathtt{TP}}$ of target points. Whenever a drone monitors a given target $i \in \Psi_{\mathtt{TP}}$, we require it to immediately communicate with its base station to deliver messages related to the ongoing monitoring activity. Such a communication may occur in a direct manner, if the target and the base station are in radio proximity with each other, or through a multi-hop routing path. Continuous connectivity to the base station ensures network responsiveness. To meet this requirement, we allow the drones of the fleet to act as both monitoring and relay devices. For this purpose, the drone trajectories may include points of the AoI where the drones can temporarily stop to act as relay nodes, in addition to the target points of $\Psi_{\mathtt{TP}}$.

We introduce the following definitions.

Definition 3.1.1. *A point $p \in \Psi_{TP}$ is said to be* inspected *by a drone u when u hovers above p for a time at least equal to the required* inspection time Γ.

Definition 3.1.2. *At a given time instant, a subset of drones $F \subseteq U$ is deployed according to a* connected coverage formation *if all the drones of F that are inspecting a target point of Ψ_{TP} are connected to their base station either directly or through a multi-hop sequence of connected drones.*

We consider the problem of designing the drone trajectories so that the drones depart and return to their base station σ and inspect the maximum number of targets along their routes by creating a sequence of connected coverage formations. More precisely we address the following problem.

Problem 3.1.1 (Optimal Connected Deployment (OCD))**.** *Given a set U of drones with a common base station σ, and available energy b, and given a set of targets Ψ_{TP} over an AoI, find trajectories for the drones of U so that they maximize the number of targets inspected by means of connected coverage formations, and break ties among equally optimal solutions by minimizing the mission completion time, i.e. the time at which the last drone returns to the base station σ.*

Theorem 3.1.1 evidences the NP-hardness of OCD which motivates the need of polynomial time heuristics.

Theorem 3.1.1. *Problem 3.1.1 (OCD) is NP-Hard.*

Proof. The generic instance of the Traveling Salesman Problem (TSP) considers an undirected, weighted graph $G = (V, E)$, and looks for a Hamiltonian cycle in the nodes of G of minimum total weight. Given such an instance, we can map it onto an instance of OCD in polynomial time. We let σ be coincident with any point of V, let it be $v \in V$, and set $\sigma = v$, while $\Psi_{\mathtt{TP}} = V \setminus \{\sigma\}$. We consider a unique drone whose battery b is larger than the maximum weight of a Hamiltonian cycle in V. For this purpose it is sufficient to set $b = |V| \cdot \max_{(i,j)\in E} w_{ij}$, where $w_{i,j}$ is the weight of edge (i, j) in the TSP instance. We then set the drone and the base station communication range $r_{\mathtt{tx}}$ equal to the maximum edge weight of the TSP instance, i.e., $r_{\mathtt{tx}} = \max_{(i,j)\in E} w_{ij}$.

Under the described setting, the unique drone of the OCD optimal solution maximizes coverage by inspecting all the points of $\Psi_{\mathtt{TP}}$. It will then break ties among all the maximum coverage solutions by looking for the one with minimum weight, which implies that any target will be visited only once, starting and returning to the base station σ. The optimal OCD solution is therefore also a Hamiltonian cycle on the points of V, with minimum weight. As the complexity of the above reduction is clearly polynomial, we derive that the OCD problem is at least as hard as the TSP problem. □

3.1.2 ILP model of the Optimal Connected Deployment

In this paragraph, we give a formulation of the OCD problem in terms of an Integer Linear Programming (ILP) model which approximates the drone trajectories by means of a fine grain discretization of the AoI, based on a regular tessellation.

Fine grain tessellation of the AoI

We hereby consider a regular tessellation of the AoI to take account of candidate relay node positions, and discretize drone trajectories accordingly. We define the set of candidate node positions as $\Psi_{\texttt{RP}}$. The set $\Psi_{\texttt{RP}}$ includes regular grid points within the convex hull of the set $\Psi_{\texttt{TP}}$. Any regular grid pattern such as a rectangular or a triangular lattice can be used here. The grid size is small enough to ensure that the distance between any two neighbor grid points is lower than or equal to $r_{\texttt{TX}}$. The definition of the set of candidate positions $\Psi_{\texttt{RP}}$ is necessary to deploy the drones in connected coverage formations, according to Definition 3.1.2.

We denote $\Psi^{+}_{\texttt{RP}} \triangleq \Psi_{\texttt{RP}} \cup \{\sigma\}$ and $\Psi^{+}_{\texttt{TP}} \triangleq \Psi_{\texttt{TP}} \cup \{\sigma\}$. A *drone trajectory* is thus a sequence of points of $\Psi^{+}_{\texttt{TP}} \cup \Psi^{+}_{\texttt{RP}}$. A trajectory point belonging to $\Psi_{\texttt{RP}}$ is either just a way-point traversed by the drone, or is a point where the drone participates as a relay in the communication between another drone and its base station.

We consider a slotted time, with loose synchronization among the drones, with time slot Δ_t. We define the *movement graph* $G_m = (V, E_m)$, where $V = \Psi^{+}_{\texttt{RP}} \cup \Psi^{+}_{\texttt{TP}}$, and $E_m = \{(i,j),\ \text{s.t. } i,j \in V,\ d(i,j) \leq d_{\texttt{max}}\}$, whose edges are the possible segments that can be traversed by a drone in a single time slot. Notice that we assume that the movement graph also contains self-loops, to consider the permanence of a drone in the same position for multiple steps when needed.

In addition to the movement graph, we define the *communication graph* $G_c = (V, E_c)$, where $E_c = \{(i,j),\ \text{s.t. } i,j \in V,\ d(i,j) \leq r_{\texttt{tx}}\}$, and $r_{\texttt{tx}}$ is the transmission range of the drones. Any pair of drones located in adjacent positions in the communication graph, can communicate with each other directly (one-hop communications). As drones are assumed to be homogeneous, we consider G_c as an undirected graph.

We denote with $\mathcal{N}_{\texttt{c}}(i)$ the set of adjacent nodes of i in the communication graph G_c, and with $\mathcal{N}_{\texttt{m}}(i)$ the set of adjacent nodes of i in the movement graph G_m.

The number of time steps of our problem formulation is upper-bounded by the number of time slots in the device battery life, i.e., $N = b/\Delta_t$. We hereby shortly denote $[N] \triangleq \{0,\ 1,\ 2, \ldots,\ N\}$.

Trajectory constraints

We introduce the binary decision variables $x^u_{ij}(n)$, for $(i,j) \in E_m$, $u \in U$ and $n \in [N]$. $x^u_{ij}(n)$ is set to 1 if drone u flies from i to j at time n, meaning that u is in i at time n and in j at time $(n+1)$. It is set to 0 otherwise. With the above variables, we define the drone trajectories according to the following constraints.

$$\sum_{k \in \mathcal{N}_{\texttt{m}}(i)} x^u_{k,i}(n) = \sum_{j \in \mathcal{N}_{\texttt{m}}(i)} x^u_{i,j}(n+1), \forall i \in V, u \in U, n \in [N], \tag{3.1}$$

$$\sum_{j \in \mathcal{N}_{\texttt{m}}(\sigma), n \in [N]} x^u_{\sigma,j}(n) = \sum_{j \in \mathcal{N}_{\texttt{m}}(\sigma), n \in [N]} x^u_{j,\sigma}(n), \forall u \in U, \tag{3.2}$$

$$\sum_{j \in \mathcal{N}_{\texttt{m}}(i), i \neq j, n \in [N], u \in U} x^u_{i,j}(n) \leq 1, \forall i \in \Psi_{\texttt{TP}}, \tag{3.3}$$

$$\sum_{i \in \mathcal{N}_{\texttt{m}}(\sigma), i \neq \sigma, n \in [N]} x^u_{i,\sigma}(n) \leq 1,\ \forall u \in U, \tag{3.4}$$

$$\sum_{j \in \mathcal{N}_{\mathtt{m}}(i)} x_{i,j}^{u}(0) = b(i), \forall i \in V,\ u \in U, \tag{3.5}$$

where $b(i)$ is an indicator function that determines whether point i is the base station σ ($b(i) = 1$) or not ($b(i) = 0$).

Equation 3.1 and 3.2 ensure that the drone traverses a closed trajectory which includes the base station σ. A drone may traverse each point of $\Psi_{\mathtt{TP}}$ only once for energy efficiency, as required by Equation 3.3. However, drones may traverse the points of $\Psi_{\mathtt{RP}}$ multiple times to realize connected coverage formations. With Equation 3.4 each drone is forced to return the base station only at the end of the mission, and once reached, it must stop there. This constraint is necessary because the following constraint (Equation 3.6) provides that drones land at the base station with no energy consumption due to hovering. Finally, Equation 3.5 considers the base station as the starting point of a drone trajectory, and allows a drone to start a delayed instant of time with respect to the others, thanks to the self-loops in G_m.

Our approach considers the energy consumption of a drone as linearly proportional to its flight time, considering both for moving and hovering activities[1].

In agreement with our energy model, we consider a constant coefficient α which reflects the average energy consumption per second of flight. We consider a limited energy availability b (expressed in energy units) at each drone $u \in U$. Then the energy constraint, due to the battery limitation of each drone, can be formulated as follows in terms of the number of time slots during which drone u flies over the AoI.

$$\sum_{(i,j) \in E_m, (i,j) \neq (0,0), n \in [N]} \alpha \cdot x_{i,j}^{u}(n) \leq b, \forall u \in U \tag{3.6}$$

Coverage constraints

We introduce the binary variable $\delta_i(n)$ for all $i \in V$ to denote the presence of a drone at point $i \in V$ at time n ($\delta_i(n) = 1$) or not ($\delta_i(n) = 0$). To relate the coverage to the trajectory variables we introduce the following constraints:

$$\delta_i(n) \geq \frac{\sum_{j \in \mathcal{N}_{\mathtt{m}}(i), u \in U} x_{i,j}^{u}(n)}{|U| \cdot |V|},\ \forall i \in V, n \in [N], \tag{3.7}$$

$$\delta_i(n) \leq \sum_{j \in \mathcal{N}_{\mathtt{m}}(i), u \in U} x_{i,j}^{u}(n),\ \forall i \in V, n \in [N]. \tag{3.8}$$

For clarity of notation, we also introduce the variable $\delta(n) \in \mathbb{N}$, defined as follows, to represent the number of covered targets, at time n:

$$\delta(n) \triangleq \sum_{k \in \Psi_{\mathtt{TP}}} \delta_k(n),\ \forall n \in [N], \tag{3.9}$$

To exploit connected coverage formations, whenever a drone $u \in U$ inspects a target $i \in \Psi_{\mathtt{TP}}$, we require it to transmit an information message to the base station. For

[1] This energy model can be extended by considering heterogeneous drones and by providing a parameterization of their operations, e.g. the energy model proposed by [88], while still adopting a linear formulation, necessary for modeling the problem as an ILP.

any two adjacent nodes i and j in the G_c graph, we define the information flow variable $f_{i,j}(n)$ at time n. For each covered target point $i \in \Psi_{\mathtt{TP}}$, the monitoring drone transmits a unit of flow. As drones may act as relay of more than one received messages, we allow the flow variables to assume values in the non negative integer field: $f_{ij}(n) \in \mathbb{N}$. Then we require:

$$f_{i,j}(n) \leq \delta_i(n) \cdot M_{\mathtt{UB}}, \forall (i,j) \in E_c, n \in [N], \tag{3.10}$$

$$f_{i,j}(n) \leq \delta_j(n) \cdot M_{\mathtt{UB}}, \forall (i,j) \in E_c, j \neq \sigma, n \in [N], \tag{3.11}$$

$$\sum_{j \in \mathcal{N}_c(i)} f_{i,j}(n) = \sum_{k \in \mathcal{N}_c(i)} f_{k,i}(n) + t(i)\delta_i(n) - b(i)\delta(n), \forall n, i \in V \tag{3.12}$$

where $M_{\mathtt{UB}}$ is an upper-bound to the amount of flow that can traverse a node at any instant of time (a valid setting for $M_{\mathtt{UB}}$ is $|\Psi_{\mathtt{TP}}|$), $t(i)$ is an indicator function that determines whether point i is a target point ($t(i) = 1$) or not ($t(i) = 0$).

Equations 3.10 and 3.11 impose a null flow between pairs of points where there are no drones, with the exception of the incoming flow to the base station. Equation 3.12 allows the information flow to traverse a connected sequence of drones taking the message to the base station. To meet these constraints the multi-hop flow variables balance each other on any node $i \in V$, with either the addition of one unit of flow if i belongs to $\Psi_{\mathtt{TP}}$ or the consumption of the flow generated by all the covered points of $\Psi_{\mathtt{TP}}$ at time n if node i is the base station.

The inspection time requirement of Γ seconds corresponds to an equivalent measure in terms of time slots: $\gamma = \Gamma/\Delta t$. To enforce persistence of the inspecting drone above the target for at least γ consecutive time slots, we consider the following constraints. Equation 3.13 ensures that for any time slot n in which a target is inspected, the drone traverses the self-loop at least γ times in the interval $[n-\gamma, n+\gamma]$:

$$\sum_{n' \in [n-\gamma, n+\gamma], 0 \leq n' < N, u \in U} x^u_{i,i}(n') \geq \gamma \cdot \delta_i(n), \ \forall n, i \in \Psi_{\mathtt{TP}} \tag{3.13}$$

Problem objective

While the primary objective of the OCD problem is coverage maximization, OCD also breaks ties by choosing the solution with minimum completion time. We introduce the decision variable $t_{\text{final}} \in \mathbb{N}$ to reflect the mission duration. t_{final} is bounded by the following constraint:

$$t_{\text{final}} \geq n \cdot \Delta t \cdot x^u_{i,j}(n), \forall u \in U, n \in [N], (i,j) \in E_m, i \neq \sigma. \tag{3.14}$$

The objective of the problem can be expressed as follows:

$$\max \sum_{i \in \Psi_{\mathtt{TP}}, j \in \mathcal{N}_{\mathtt{m}}(i), i \neq j, n \in [N], u \in U} x^u_{i,j}(n) - \varepsilon \cdot t_{\text{final}}, \tag{3.15}$$

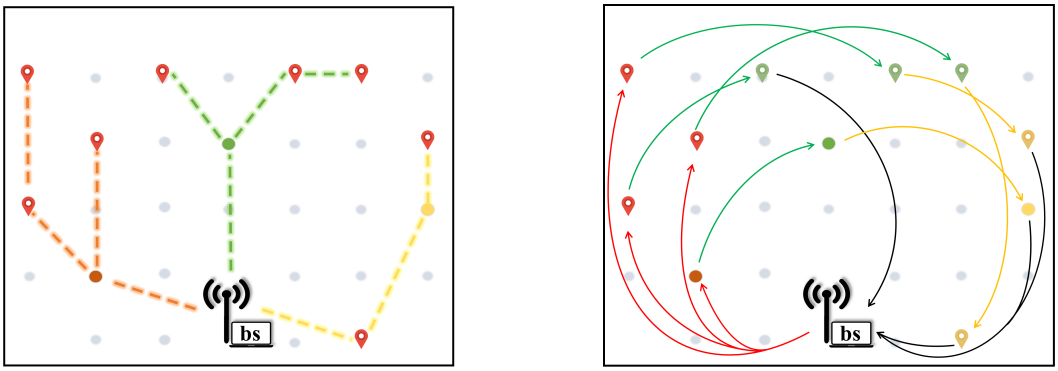
where we use a linear combination of the two objectives, namely coverage and completion time, assigning an arbitrarily small, but positive weight ε to completion time, in order to use it only to distinguish equally optimal solutions for the coverage objective.

3.1.3 Greedy Connected Deployment (GCD)

To efficiently address the OCD problem, we introduce a polynomial-time greedy approach hereafter called *Greedy Connected Deployment (GCD)*.
The proposed algorithm works in two phases:

1. ***Deployment Trees Generation*** - It designs connected formations of drones (also referred to as *deployment trees*). Figure 3.1(a) shows an example of three formations covering 8 targets, with 4, 4, and 3 drones, respectively.

2. ***Deployment Trees Scheduling*** - It determines the cost for the fleet to move from one formation to another, and looks for a formation schedule of minimum cost. Figure 3.1(b) shows how the three formations can be scheduled by highlighting the trajectories of individual drones.



(a) Phase 1 (b) Phase 2

Figure 3.1. GCD Example

Phase 1: Deployment Trees Generation

Phase 1 of GCD consists in constructing *deployment trees*, i.e. drone formations connected to the base station σ. The nodes of a deployment tree represent the position drones should reach to cover target nodes, while staying connected with the base station in a multi-hop manner. Any deployment tree τ is rooted at the base station σ. The leaves of a deployment tree are target points of $\Psi_{\texttt{TP}}$, hereafter denoted as $L(\tau) \subseteq \Psi_{\texttt{TP}}$. The intermediate nodes can either be relay or target points, allowing multi-hop connection with the base station, we refer to them as $I(\tau) \subseteq \Psi_{\texttt{TP}} \cup \Psi_{\texttt{RP}}$. Every node of a deployment tree is a drone position in the corresponding formation, with the exception of the root, which is the base station, i.e., $\tau = (V_\tau, E_\tau)$, with $|V_\tau| \leq |U| + 1$. As the available drones U may be insufficient to cover the entire set of targets in a unique connected formation, the goal of GCD phase 1, is to design a set of deployment trees $\mathcal{T}$, that covers all the target nodes. In this phase, the deployment trees are created starting from a $|U|$-Bounded Depth Steiner Tree [89] defined on the communication graph G_c and rooted at the base station σ, whose terminals are the target points of $\Psi_{\texttt{TP}}$. More specifically, we calculate a tree τ_{full}

Algorithm 4: Greedy Connected Deployment (GCD) - Phase 1

Input: $G_c : (V : (\{\sigma\} \cup \Psi_{\texttt{TP}} \cup \Psi_{\texttt{RP}}), E_c)$ com. graph, U drones set
Output: $\mathcal{T}$ deployment trees set

1 $\widehat{\Psi}_{\texttt{TP}} \leftarrow \{\sigma\}$
2 $\tau_{\text{full}} : (V_{\text{full}}, E_{\text{full}}) \leftarrow$ bd-steiner-tree $(G_c, \Psi_{\texttt{TP}} \cup \{\sigma\}, |U|)$
3 $\tau_{\text{opt}} : (V_{\text{opt}}, E_{\text{opt}}) \leftarrow \emptyset$
4 **while** $\widehat{\Psi}_{\texttt{TP}} \neq \Psi^{+}_{\texttt{TP}}$ **do**
5 $\quad \psi_{\text{best}}, \tau_{\text{best}}, c_{\text{best}} \leftarrow \emptyset, \emptyset, \infty$
6 $\quad$ **for** $\psi_{\text{temp}} \in \Psi_{\texttt{TP}} \setminus \widehat{\Psi}_{\texttt{TP}}$ **do**
7 $\quad\quad \tau_{\text{temp}} : (V_{\text{temp}}, E_{\text{temp}}) \leftarrow \tau_{\text{opt}} \cup P(\tau_{\text{full}}, \sigma, \psi_{\text{temp}})$
8 $\quad\quad$ **if** $|V_{\text{temp}}| - 1 \leq |U|$ **and** $|E_{\text{temp}}| < c_{\text{best}}$ **then**
9 $\quad\quad\quad \psi_{\text{best}}, \tau_{\text{best}}, c_{\text{best}} \leftarrow \psi_{\text{temp}}, \tau_{\text{temp}}, |E_{\text{temp}}|$
10 $\quad$ **if** $\tau_{\text{best}} \neq \emptyset$ **then**
11 $\quad\quad \tau_{\text{opt}} \leftarrow \tau_{\text{best}}$
12 $\quad\quad \widehat{\Psi}_{\texttt{TP}} \leftarrow \widehat{\Psi}_{\texttt{TP}} \cup \{\psi_{\text{best}}\}$
13 $\quad$ **else**
14 $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_{\text{opt}}\}$
15 $\quad\quad \tau_{\text{opt}} \leftarrow \emptyset$
16 **if** $\tau_{\text{opt}} \neq \emptyset$ **then** $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_{\text{opt}}\}$
17 **return** $\mathcal{T}$

that includes all the targets of $\Psi_{\texttt{TP}}$, a minimum number of edges and other nodes of G_c, under the constraint that the depth of the tree be lower than or equal to $|U|$. Such a tree $\tau_{\text{full}} = (V_{\text{full}}, E_{\text{full}})$ is a feasible solution for our deployment problem only if $|U| \geq |V_{\text{full}}| - 1$ nodes (we recall that as the root σ does not require a positioned drone). To address the general case where $|U| < |V_{\text{full}}| - 1$ we propose Algorithm 4.

The input is the communication graph G_c and the set of drones U. We assume that all the nodes of the communication graph can be reached from the base station, through a multi-hop path of length lower than or equal to $|U|$. More precisely, it must be: $|P(G_c, \sigma, \psi)| - 1 \leq |U|$, $\forall \psi \in \Psi_{\texttt{TP}}$, with $P(\cdot)$ being the shortest path function. In the initialization phase (**lines 1-3**), we let: $\widehat{\Psi}_{\texttt{TP}}$ be the set of nodes that have been covered so far; τ_{full} be a Steiner tree that spans the whole set of target nodes and σ, of depth at most $|U|$; and τ_{opt} be an empty candidate deployment tree to be added to $\mathcal{T}$, once fully grown. Until the whole set of targets is covered i.e., $\Psi_{\texttt{TP}} = \widehat{\Psi}_{\texttt{TP}}$, the algorithm tries to add candidate deployment trees τ_{opt} to $\mathcal{T}$ (**lines 4-9**). These trees are iteratively grown in a greedy manner, for every target ψ_{temp} yet to cover in $\Psi_{\texttt{TP}} \setminus \widehat{\Psi}_{\texttt{TP}}$, it is checked whether the path spanning ψ_{temp} and the base station in τ_{full} (i.e., $P(\tau_{\text{full}}, \sigma, \psi_{\text{temp}})$) should be added to τ_{opt} or not. Two checks are made, one is whether the addition to τ_{opt} can be made without exceeding the number of available drones; the other is whether the path is the shortest among all the possible paths.

If both checks go through (**lines 10-15**), then a new extension of τ_{opt} is found and τ_{opt} is set to the new grown version τ_{best}. The new reached target node is added to the set of covered nodes $\widehat{\Psi}_{\texttt{TP}}$. In case no new target node could be added, without exceeding the number of available drones, the candidate deployment tree τ_{opt} is added to the cover $\mathcal{T}$ and reset. Finally, (**lines 16-17**), the last candidate deployment tree τ_{opt} is added to the cover and the algorithm outputs the set of deployment trees $\mathcal{T}$ that includes all the targets.

Algorithm 5: Greedy Connected Deployment (GCD) - Phase 2

Input: $\mathcal{T}$ deployment trees set
Output: $\mathcal{S}^*$ an ordered list of deployment trees

$\tau_\sigma \leftarrow (\{\sigma\}, \emptyset)$
$G_{\mathcal{T}} \leftarrow (\mathcal{T} \cup \{\tau_\sigma\}, \{(\tau, \tau', c_m(\tau, \tau')) \mid \tau, \tau' \in \mathcal{T} \cup \{\tau_\sigma\}\})$
$\mathcal{S} \leftarrow \text{TSP}\ (G_{\mathcal{T}}, \tau_\sigma)$
$\mathcal{S}^*, p^* \leftarrow (\emptyset, 0)$
for $\tau_i, \tau_j \in \mathcal{S} \mid 1 < i \leq j \leq |\mathcal{S}|$ **do**
  $\mathcal{S}', p' \leftarrow \left(\langle \tau_\sigma, \tau_i, \tau_{i+1}, \ldots, \tau_{j-1}, \tau_j, \tau_\sigma \rangle, \left| \bigcup_{k=i}^{j} L(\tau_k) \right| \right)$
  $c' \leftarrow c_m(\tau_\sigma, \tau_i) + \sum_{k=i}^{j-1} c_m(\tau_k, \tau_{k+1}) + c_m(\tau_j, \tau_\sigma)$
  if $\alpha c' \leq b$ **and** $p' > p^*$ **then**
    $\mathcal{S}^*, p^* \leftarrow (\mathcal{S}', p')$
return $\mathcal{S}^*$

Phase 2: Deployment Trees Scheduling

Phase 2 of GCD consists in finding an optimal schedule of deployment trees, and is addressed by Algorithm 5.

The choice of such a schedule considers the effort for the entire fleet of drones to execute the transition. We will refer to this effort as the cost of migration $c_m(\tau, \tau')$ with τ being the source deployment tree, and τ' the destination. Intuitively, such a cost function is the *time* required for the drones covering τ, to establish a connection between $L(\tau')$ and σ according to τ'. We consider assignments that minimize the maximum transition time of the drones, because we want nodes in $L(\tau')$ to be connected as soon as possible. We then reduce the problem to what in the literature in known as the Bottleneck Matching Problem (BMP) for bipartite graphs [90]. We recall that BMP takes a bipartite graph $G = (V_0, V_1, E)$, and a non-negative edge cost function $c(e) > 0$, $\forall e \in E$, and finds a perfect matching $M \in M_p$ where M_p is the set of subsets of E of cardinality $|V_0| = |V_1|$, of minimal cost, i.e., $\min_{M \in M_p} \max_{e \in M} c(e)$. We define our reduction to a BMP as follows. Let $G_{(\tau,\tau')} = (V_\tau, V_{\tau'}, E_{(\tau,\tau')})$ be the complete bipartite graph such that, nodes in part V_τ represent nodes in the source deployment tree τ, while nodes in part $V_{\tau'}$ represent nodes in the destination deployment tree τ'. As these parts represent drones deployments, we add to V_τ as many replicas of the base station as the number of unused drones in τ, we do the same with $V_{\tau'}$. Eventually we have that $|V_\tau| = |V_{\tau'}| = |U|$. The set of undirected edges $E_{(\tau,\tau')} \subset V_\tau \times V_{\tau'}$ are weighted with $d(u, w)/v$, $\forall u \in V_\tau, w \in V_{\tau'}$, i.e., the time needed for the drone to move from node u to w. Solving a BMP instance with $G_{(\tau,\tau')}$ as input, will eventually produce a perfect matching $M^*(\tau, \tau')$ that is the migration plan set for all the drones in the fleet. The transition cost is then computed as $c_m(\tau, \tau') = \max_{e \in M^*(\tau,\tau')} c(e)$.

Having the estimated effort needed to migrate from one deployment tree to another, we can determine an optimal route for the drones, which starting from the deployment configuration τ_σ (i.e. all the drones are on the base station), visit all the trees and go back to τ_σ. This is clearly a Hamiltonian cycle one can find by solving the Traveling Salesman Problem (TSP). We reduce the problem of finding an optimal migration schedule to the TSP as follows. Let $V_{\mathcal{T}} = (\mathcal{T} \cup \{\tau_\sigma\})$,

such that $V_{\mathcal{T}}$ is the set of nodes representing deployment trees in $\mathcal{T} \cup \{\tau_\sigma\}$. Let $G_{\mathcal{T}} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ be a complete, undirected graph, where all edges in $E_{\mathcal{T}}$ are weighted with $c_m(\tau, \tau')$, $\forall \tau, \tau' \in E_{\mathcal{T}}$. Solving a TSP instance with $G_{\mathcal{T}}$ as input, will produce a scheduled migration that is optimal. The algorithm creates $G_{\mathcal{T}}$, input of the TSP (**lines 1-3**), which returns a list of deployment trees $\mathcal{S}$, that is a migration schedule. Such a schedule is optimal if the drones are hypothetically assumed to have infinite battery life. As in a realistic scenario the battery life is a limiting factor on the cruise time of the drones, we introduce a TSP tour cutting technique (**lines 4-10**). First the energy expenditure c' of the fleet, under a given sub-tour $\mathcal{S}'$, is computed as the sum of the maximum cost of migrating from a deployment tree in $\mathcal{S}'$ to the next. Finally the algorithm returns the sub-tour $\mathcal{S}^*$, i.e. the tour that allows to visit the highest number of targets p^*, that fits the battery constraint of the drones.

Properties of GCD

Theorem 3.1.2 (Time Complexity of GCD)**.** *GCD with input G_c and U has polynomial time complexity $O(|V|^{4.5})$.*

Proof sketch. We consider the two phases of GCD. The complexity of phase 1 is: $O(|V|^3 + |\Psi_{\mathtt{TP}}|^2 \log(|V|))$. The computation time for the approximated BDSTP is $O(|V|^3)$, [89,91], with constant 1.52 approximation. The while loop is executed until the set of reachable targets is fully covered, in $O(|\Psi_{\mathtt{TP}}|)$ iterations. In particular, one target node is covered every two while loop iterations in the worst case. In fact, if no new target can be covered in an iteration, as it would require more drones than available, the algorithm would create a new deployment tree, and would therefore enable coverage of a new target at the next iteration. At each while iteration, a simple path lookup in τ_{full} is done with cost $O(\log |V|)$, for every uncovered target in $|\Psi_{\mathtt{TP}}|$.

The complexity of phase 2 is: $O(|\Psi_{\mathtt{TP}}|^3 + |\Psi_{\mathtt{TP}}|^2(|U|^2\sqrt{|U|/\log|U|} + \log|\Psi_{\mathtt{TP}}|))$. The maximum number of deployment trees is $|\Psi_{\mathtt{TP}}|$, for every pair of deployment trees $|\Psi_{\mathtt{TP}}|(|\Psi_{\mathtt{TP}}| - 1)/2 = O(|\Psi_{\mathtt{TP}}|^2)$, a migration cost is evaluated, which consists into solving a BMP of a complete, undirected, bipartite graph with $|U|$ nodes. Such a BMP can be computed in $O(|U|^2\sqrt{|U|/\log|U|})$ [90]. We consider the Christofides approximation to the TSP solution [15], with complexity $O(|\Psi_{\mathtt{TP}}|^2 \log|\Psi_{\mathtt{TP}}|)$. Finally, the algorithm selects the longest sub-tour in the number of nodes, from the TSP solution. The computation and evaluation of all the sub-tours from the TSP trajectory costs $O(|\Psi_{\mathtt{TP}}|^3)$. Considering that $|E| = O(|V|^2)$ and $|U| = O(|V|)$, with simple algebraic steps we obtain that the overall complexity:

$$O\left(|V|^4\sqrt{\frac{|V|}{\log|V|}} + |V|^3 + |V|^2\log|V|\right) = O(|V|^{4.5}). \qquad \square$$

3.1.4 Performance Evaluation

In this section we evaluate GCD against the optimal solution and two state of the art approaches, both in a simulated environment and through real-field experiments. In the *simulated environment*, we consider 2 different scenarios: (1) In the first scenario we deploy 5 drones and let the number of target nodes vary from 3 to 15,

and compare GCD to OCD. (2) In the second scenario, we deploy 10 drones and let the number of target nodes vary from 10 to 40, to study the performance of GCD with respect to state-of-art approaches. As for the *real-field experiments*, we deploy 4 drones with a number of targets varying from 8 to 15.

As benchmarks for comparisons, we consider two state-of-arts solutions for multiple drones and multiple robots target inspection. We first consider *AC-GaP* (See Section 2.1), a recent algorithm for multiple drones that aims at minimizing the target average inspection delays, considering multiple trips for the drones, with recharging and off-loading operations in between. We recall that, AC-GaP computes a set of feasible trajectories, traversing different subsets of the targets, and then it greedily selects and schedules some of them, according the drones' battery constraints. To cope with our problem setting, we consider a single trip, and we consider targets as visited only if the drones are able to communicate with the base station during the inspection. As second benchmark we consider *2S-APX* [77], a work for multiple robots that provides inspection with recurrent connectivity (i.e., each visited target requires a multi-hop path toward the base station to send the data). 2S-APX iteratively solves the inspection problem by means of a two-stage strategy. The first stage computes the next set of drones' locations to maximize the number of inspected targets while maintaining connectivity with the base-station. It enumerates and evaluates the subsets of targets, to compute the cheapest Steiner tree allowing to connect targets to the base station. The second stage decides which robot goes to which vertex by applying the Hungarian assignment algorithm, which minimizes the cumulative traveled distance. As *2S-APX* does not consider energy consumption, which is a critical aspect for drones, to cope with our scenario, we cut the mission as soon as the battery is depleted.

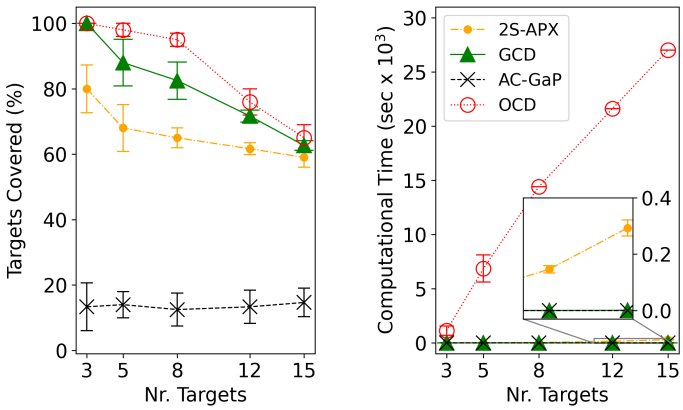
We evaluate the algorithms in terms of several performance metrics. We consider the *Percentage of Covered Targets*, during the mission, where a target is considered as covered only if there exists a connection toward the base station during its inspection. We then consider *Cumulative Target Inspection*, which reflects the integral of the former, and reflects early coverage by providing a time based cumulative evaluation of the percentage of covered targets. We also consider the *Average Traveled Distance*, which reflects the average distance traveled by all the drones during the mission. Finally, we evaluate the algorithms in terms of *Computation Time*.

Simulated Environment

We run our tests on a Lenovo X3550 M5, with CPUs Intel(R) XEON(R) E5-2650@ 2.20GHz, 16 cores and 32 GB RAM [68]. For OCD we used the Gurobi solver [30].

We simulate commercial drones with a communication range of $100\,m$, a speed of $5\ m/s$; 5 seconds of inspection time; and a fixed base station placed at the bottom of the area of interest (see Figure 3.1). While OCD and GCD work independently of the specific energy model, we consider a simplified energy consumption model, i.e., we consider a constant coefficient $\alpha = 1$ which reflects the average energy consumption per second of flight (see Section 3.1.2).

For each point in the plots we execute 30 runs, and the error bars represent the standard error.



(a) Perc. of Covered Targets (b) Computation Time

Figure 3.2. First Scenario.

First Scenario The first simulations consider all the algorithms in a scenario with a field of $300 \times 300\,m$ and 5 drones with $150s$ of flight time. This simplistic setting is needed to create a scenario where OCD can be computed in a moderate execution time. We vary the number of targets from 3 to 15.

In Figure 3.2(a) we show the percentage of covered targets. The plot shows how OCD has the best performance visiting almost all the targets, under the drones battery limitations. Our greedy proposal GCD performs close to OCD. It visits at least 90% of the targets visited by the optimal in the worst case (i.e., 8 targets), while 2S-APX visits 30% fewer targets than OCD, in the same scenario. As AC-GaP does not require connectivity to the base station, therefore only targets in the communication range of the base station are correctly inspected, and only in lucky circumstances it finds a multi-hop path for communications with the base station.

Figure 3.2(b) shows that the optimal performance of OPT comes at the expense of computation time. OCD requires $25.5 \cdot 10^3\,s \approx 7hrs$ when the number of targets is 15, a time that cannot be accepted in emergency critical scenarios.

Second Scenario In the second simulation scenario, we consider a larger area of $1500 \times 900\,m$ with 10 drones whose battery allows 600 seconds of flight time. We investigate the performance of the algorithms by varying the number of targets from

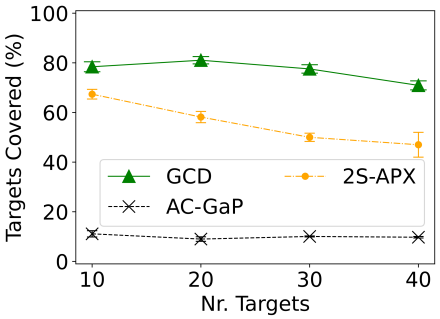


Figure 3.3. Perc. of Covered Targets

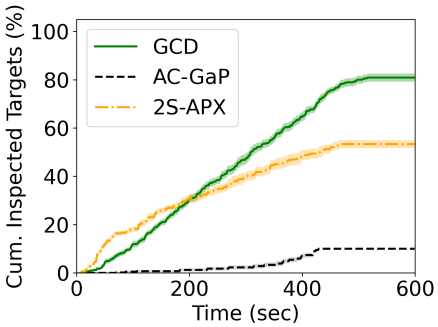


Figure 3.4. Cum. Targets Inspection

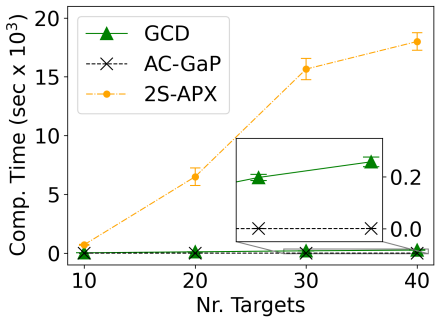


Figure 3.5. Computation Time

Figure 3.6. DJI F-550

10 to 40. Due to the extremely high computation time of OCD in this setting, we only compare GCD against AC-GaP and 2S-APX algorithms.

Figure 3.3 shows the percentage of covered targets with 10 drones, increasing number of target nodes. GCD shows the best performance with an improvement from 10% to 40% with respect to 2S-APX.

In Figure 3.4 we analyze the case of 30 targets. Not only GCD outperforms 2S-APX in terms of the total number of visited targets, but it also visits about 50% of the targets in the first 350 seconds of the mission, while 2S-APX requires 450 seconds. Instead, AC-GaP visits only 20% of the targets visited by GCD, which are almost all in the communication range of the base station.

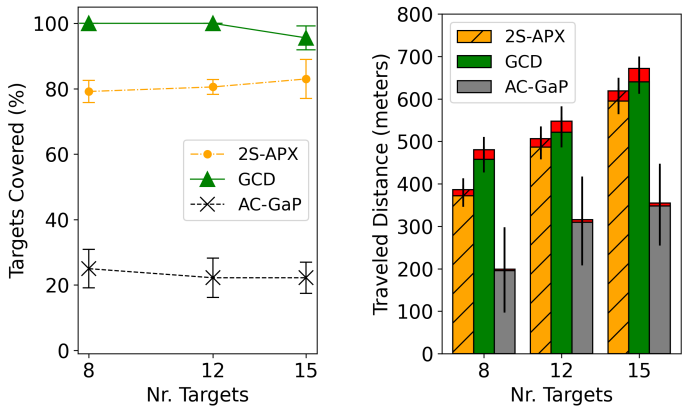
Finally, in Figure 3.5 we notice how the computation time for 2S-APX is a serious issue for the applicability of the method in an emergency critical scenario. In fact, the computation time of 2S-APX grows significantly when the number of targets increases (i.e., from 5 minutes to 5 hours with 10 and 40 targets respectively).

Real-Field Experiments

In this section we experimentally investigate the performance of GCD and validate the simulation results in a real scenario.

The test-bed includes a fleet of 4 DJI Flame Wheel (F550) [55] with GPS and communication capabilities, and a laptop to setup the deployment and act as a base station. While the design and implementation of collision avoidance mechanisms is beyond the scope of this work, we recall that our trajectories are computed offline and thus collisions may be easily prevented by using different flight heights, for the drones whose trajectories intersect one another. Moreover, in the presence of physical obstacles in the area, drones may be equipped with on-board collision avoidance systems (e.g., DJI Phantom 4 Pro V2.0 [92]).

For the multi-hop communication purpose we equipped each drone with a Raspberry Pi Zero W [93], with a WiFi antenna, which weighs less than 10 grams. We implemented the B.A.T.M.A.N. routing protocol [94] to route packets towards the base-station. We notice that, B.A.T.M.A.N. is able to automatically handle disconnections and re-connections of the drones, by means of periodic hello packets which are known as originator messages (OGM). Thus, it is able to dynamically manage data exchange.



(a) Perc. of Covered Targets (b) Average Traveled Distance

Figure 3.7. Real-Field Experiments

The experiment location has a size of $300 \times 200\,m$ with no obstacles. The drone speed was limited by $4\,m/s$ and, by prior in field measurements, we consider $100\,m$ of communication range and around 240 seconds of flight time. All the drones were equipped with LIPO batteries, 3500 mAh, and the experiments were performed under good weather conditions, with a wind speed less than $1m/s$ and a temperature around 15°C. We evaluate the algorithms by varying the number of targets from 8 to 15, and we consider 5 execution runs to take average performance measurements.

In Figure 3.7(a) and 3.7(b) we evaluate the performance in the real-field and we validate the simulations. Figure 3.7(a) validates the trends of the first scenario. In particular, GCD outperforms both 2S-APX and AC-GaP in terms of target coverage, of about $15 - 20\%$ and 70% respectively.

Figure 3.7(b) shows the average distance traveled by the drones during the mission. It shows how GCD utilizes the drone resources better than the other algorithms. With GCD the drones travel more than with 2S-APX and AC-GaP, but they also cover more targets. Instead, 2S-APX is not able to benefit from the available energy, and results in lower traveled distance, but also in a lower number of visited targets. In the same setting, AC-GaP shows the worse performance, as the drones do not facilitate packet delivery with their movements. Finally, the red bars in the figure highlights the differences between the traveled distance estimated in the simulations, and the actual distance traveled in the real field. Is worth to notice that, conservative parameters are used in the simulation to avoid unfeasible trajectories and therefore, the traveled distance in the real-field was slightly lower than the one computed in the simulations.

3.1.5 Conclusion

We present the connected deployment problem, where a FANET creates connected formations to ensure multi-hop high-rate communications while performing a monitoring task in an area of interest. We show that this problem is NP-hard, and we provide an optimal formulation and a polynomial time solution, called Greedy Con-

nected Deployment (GCD) based on a two phase approximation of the problem. We demonstrate polynomial time complexity of GCD and we study its performance both in a simulated environment and real field. Results show that GCD performs close to the optimal solution, and outperforms existing solutions in the literature, with remarkably lower computation time making it particularly adequate for emergency critical scenarios.

3.2 SIDE: Self drIving DronEs embrace uncertainty

In the previous sections we demonstrated how fleets of multiple and cooperative drones are particularly suitable for scenarios where a timely and reliable intervention is of uttermost importance, including post disaster operations. Nevertheless, a number of issues need to be addressed before autonomous fleets of drones can be efficiently used [7] also in dynamic environments. In particular, the drones of the fleet require ad-hoc coordination and control systems which should be able to: determine tasks, plan drone trajectories across the field of interest, and update the mission strategy based on local findings along the field. Moreover, self-coordination capabilities are essential to take the human out of the control loop as much as possible, to guarantee a quickly deployable, highly responsive monitoring network.

Existing issues. Prior research has addressed the problem of inspecting target locations in a field by means of multiple vehicles in a centralized manner [16–19,36,43, 48,49,95]. However these works cannot be used in dynamic environments, since they assume perfect knowledge of target locations and persistence of the related events during the monitoring activity [95]. Indeed, many scenarios require the capability to detect and inspect dynamic events whose location and duration can only be estimated with non negligible *uncertainty*, i.e., events may arise any time during the mission, and their positions are known in a probabilistic manner. As an example, a squad of drones employed for search and rescue in a post-disaster setting may identify collapsed buildings, which may potentially host many subsequent critical events (e.g., presence of survivors, further collapses, or new accidents). In crime prevention and monitoring applications, drones may facilitate the identification of dangerous areas where future crimes are likely to occur in the proximity of prior events [96]. A squad of drones may also be helpful in maritime traffic accidents monitoring, where the drones may identify hazardous locations for maritime traffic, or on-going accidents in need of further intervention [97].

Approach. In this work we target scenarios with dynamic, temporally and geographically correlated events. In particular, we consider the problem of monitoring dynamic events with minimum inspection latency, under uncertainty of time and location of event occurrences. Under our approach, while exploring the area of interest, the drones of the squad build and update a shared map that provides a probabilistic representation of the ongoing events. The management of this map is realized by applying the Parzen-Rosenblatt approach, also called the multivariate kernel density estimation (KDE) method. The use of this map enables the squad to determine the most likely locations of future events within the field, during the mission execution. Individual drone trajectories are then calculated using a virtual force model which guides drone movements. In particular, drones are attracted towards critical zones of their map, while they exert mutually repulsive forces to ensure load balancing and collision avoidance.

We consider battery-powered drones that use home depot locations for take-off and landing operations, battery replacement, data offloading and map updates, upon need. The drones of the fleet execute several consecutive flights, during which they inspect targets within the field of interest, and possibly detect the presence of new ongoing events requiring additional monitoring. The drones create a highly

responsive self-adaptive network, with self-healing capabilities. Simulations and test-bed experiments show that our approach outperforms previous solutions under a wide range of experimental settings.

Summary of contributions. The original contributions of the work are the following:

- We propose a distributed algorithm which lets drones build and update a dynamic probabilistic map of the events. Such a map is managed by means of the multivariate kernel density estimation (KDE) method, also known as the Parzen-Rosenblatt approach.
- We contribute a distributed trajectory planning algorithm that employs a virtual force approach to guide drone movements along the Area of Interest (AoI), while taking account of upcoming events and their dynamics.
- We evaluate our proposal through extensive simulations in terms of key performance metrics including average inspection delay, coverage percentage, and average delay in new event detection. We compare our approach to two previous proposals showing that it significantly outperforms the others in all the relevant metrics, doubling the inspection rate and halving the inspection latency in many of the considered scenarios.
- We perform real-field experiments to validate and confirm the analysis carried out via simulations.

3.2.1 Related Work

As our proposal considers a path planning algorithm with a collaborative map built through multi-hop communications, we first report some surveys and tutorials on UAV networks, and then we present the related work for path planning and collaborative map building.

UAV Networks

In the last years, networks of UAVs have been increasingly used in several scenarios to: offload users' tasks and data to mobile edge servers [98]; enhance ground user connectivity, acting like aerial base stations [99]; capture and stream real time videos [100]; and collect data from ground sensors [101].
To make UAV networks practical, a number of communication issues have to be addressed [7]. Recent literature studies the performance of previously proposed protocols in the context of aerial communication and networking [9, 11] but, to date, a de facto standard does not exist [12]. Concerning routing, the most promising approach leverage the geographical scheme [102], or controlled mobility to deliver packets [24]. Concerning data link, several air-to-ground (AG) and air-to-air (AA) propagation channel models have recently been studied to support the design of communication protocols for UAVs networks [103–105]. However, AA communications between UAVs have not been extensively studied, and the free space model is still the state of the art [104].

In our work, we do not rely on a specific routing algorithm, as we consider message broadcast to neighbors nodes, while we adopt a two ray propagation model [106], which extends the commonly used free space model for AA communications.

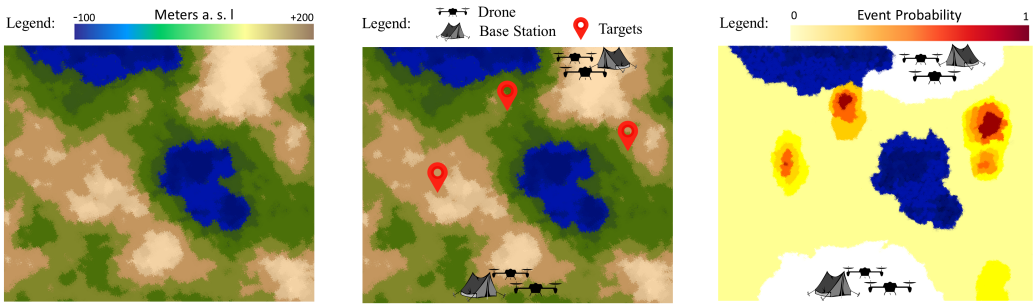
Path Planning

The problem of inspecting target locations in a field through multiple vehicles has been addressed with several centralized approaches including graph algorithms [16,17,36,38–40], optimization problems [18,19,95], genetic [45] and machine learning algorithms [47, 48, 99, 107], and virtual force approaches [108]. All of the above aim at designing multiple trajectories satisfying monitoring, inspection delay and energy requirements, while visiting a set of points of interest in an area. However, these works assume accurate knowledge of the area of interest or a static set of targets, and cannot be directly applied to search operations with unknown event locations. For its analogy with our work in the objective to minimize the target inspection delay, we considered *AC-GaP* (see Section 2.1) as benchmark for the performance comparisons. The proposed approach determines the trajectories of a multi-trip mission. We modified it so as to dynamically update the set of target points at each trip, upon detection of new events by the drones.

Few works specifically focus on search-and-reconnaissance operations for UAVs with a single drone [109], and multi-drone systems [110]. Unlike our work, the uncertainty map is given at the beginning of the mission, thus the drones only visit the uncertain zones to increase their awareness of the area but they do not consider the event dynamics.

The works in [111, 112] optimize target coverage and cooperation among drones in dynamic environments where events, including drone failures or threats in the area of interest, can appear during the mission. In particular, [112] addresses the path planning problem by maximizing the number of targets visited at most once, before the drone batteries deplete. However, both the proposals are centralized and do not provide full coverage of the area of interest. In addition, notification of new events requires external sources of information.

We observe that only few works on mission control and cooperative search for ground mobile robots can be adapted to our scenario [51–53], while most of the other works for ground robots differ in assumptions and addressed scenarios. In particular, differently from mobile sensors or terrestrial robots, aerial drones do not benefit from a static deployment (i.e., they consume energy while hovering) and, moreover, they take advantage of expanding coverage thanks to device re-positioning. For these reasons, most of the existing work can't be applied in our scenario.
We introduce an approach called *Sweep*, inspired by a related approach designed for terrestrial robots [53], and adapted to our scenario by performing multiple trips within drones battery constraints. The main goal of *Sweep* is to obtain complete area monitoring coverage, and it is used in Section 3.2.5 as another baseline for performance comparisons with our proposal.



a) Input Map b) Search Mission with Targets c) Search Mission with Uncertainty Map

Figure 3.8. Example of a common target-based search mission (b) against a search mission with uncertainty of target positions (c).

Collaborative map building

In our proposal, the drones share their observation of the environment to gradually refine and build a probabilistic map of ongoing events. The problem of combining chunks of information from several drones to build a single robust view of the environment, has been previously studied under different scenarios [113–120]. A first scenario involves robots that create a map in a common reference frame, assuming that every robots knows its position. In the seminal work [120] robots independently build their maps, and fuse them with the global map using the Kalman Filter. A second scenario considers relative positions of drones: the drones meet and localize themselves in each other's maps to build the global map. In [113], drones consider only relative positions, exchange data to build a global map with limited communication capabilities, and localize themselves in each other's maps during the merging phase to correctly build the global map. Finally, a third scenario assumes that the robots relative positions are unknown, and the map is built without positional information. To address such a challenging scenario [114] defines a similarity metric for the local maps, and a stochastic search algorithm, to fuse them in the global map.
In our work, each drone knows its exact relative positions and creates a map in a common reference frame, using the Parzen-Rosenblatt window method [121] [122]. Therefore, building the collaborative map does not require any sophisticated algorithm. However, in case positioning is not available or the map frame is unknown, one of the above works can be easily integrated in our framework.

To conclude, we note that both the Kernel Density Estimation (KDE) and Virtual Force (VF) approaches are well-known methods and not original contributions of our work. The originality of our approach resides in the use of KDE to dynamically modulate the intensity and direction of the attracting forces that guide the movement of the drones along the area of interest. Without the KDE approach to estimate the geographic map of events and their dynamics, the VF alone would only provide a swarm deployment to cover the area of interest without focusing on the ongoing events and their dynamics.

3.2.2 Path Planning under uncertainty

Our problem formulation addresses a scenario in which a squad of aerial drones aims at inspecting an Area of Interest (AoI) with several dynamic events. Most

importantly, we consider possible *uncertainty* in the mission requirements, i.e., *events may arise any time during the mission, and their positions are known in a probabilistic manner*. Therefore, the apriori knowledge of the event's location can be represented as an uncertainty map, which is a function of the probability distribution of the targets in the area. The highest the probability associated to an area, the greater the chance that area will host an interesting event.

Figure 3.8.a shows an example of an area of interest to monitor, with different heights above sea level. Figure 3.8.b shows the same area in a deterministic scenario, with four drones and three known targets. These points may be the known locations of survivors of a catastrophe, requiring medicines or water to be dropped from aerial drones, or more in general areas requiring immediate surveillance and local inspection. Figure 3.8.c represents the same area in case of uncertainty. In this case only most critical zones can be identified (e.g., collapsed buildings), and they can potentially host subsequent critical events (e.g. presence of survivors, further collapses). Different zones in the area are labeled with their probability to host a critical event, from white (i.e., safe zone) to dark red (i.e., extremely likely to find an event). Notice that, the third scenario generalizes the problem with known targets: the probabilistic map can easily incorporate known targets using zones with probability equal to 1.

Events and Drones

Throughout this work we consider a simplified model of the drone devices. Far from narrowing the applicability of the proposed approach, this simplified model serves the purpose of providing a general network architecture and a methodological framework to deal with uncertain knowledge of upcoming events of interest. We define an event of interest as any location requiring immediate surveillance and local inspection. Our model can take more peculiar device specifics into account, replacing the assumptions described in this section.

We consider a squad of homogeneous battery-powered aerial drones, with computational and communication capabilities, and a set of base-stations. Drones are equipped with a Global Positioning System (GPS)[2], a communication unit, and an on-board CPU for data processing and movement control. Drones have limited battery power, requiring maintenance to carry on and complete the mission (i.e., battery recharging or replacements). In our approach we consider a simplified energy consumption model, where each drone has a linear proportional consumption to its flight time, considering both hovering and monitoring activities. Notice that, commercial drones often define battery lifetime in term of flight-time, nevertheless, our proposed solution can be easily adopted with any energy consumption model (e.g., the energy model proposed by [63].)

The mission can be executed in critical or unsafe scenarios (e.g., the aftermath of an earthquake or a disaster), in which fixed cellular infrastructure may be broken or completely knocked down, making direct communication among devices impossible [125]. Therefore, we assume that drones and base stations can only use short-range radios (e.g., WiFi) and communicate through a multi-hop communication.

[2]In case GPS is not available existing alternative techniques can be used [123] [124] to provide localization.

Drones approach the base-station to offload collected data through the wireless link, and they occasionally land for maintenance and battery replacement.

During the mission drones fly at approximately constant height above the ground, to monitor the area and collect accurate data. For example, considering drones endowed with a camera, the mission requires a given Ground Sampling Distance (GSD), which defines the distance between two consecutive pixel centers measured on the ground, and controls the visible details on the image. The GSD is a function of the sensor width, focal length of the camera, pixels, and the height of the drone [126]. Thus, the drones have to fly at a given altitude above the ground to achieve the required GSD, considering their homogeneous cameras. Nevertheless, as discussed in the rest of the work, during landing phases, or to avoid collisions, drones can adjust their altitude.

Different from existing works, we define two different monitoring activities, *event detection* and *event inspection*, possibly involving different sensory equipment with different features. As a practical example, let us consider monitoring eruptions in a geyser region. A drone can easily detect the occurrence of an eruption by using its onboard camera or microphone from a distant location. Nevertheless, to measure temperature and gas composition, the drone resorts to a sensory equipment which requires a closer inspection. Without loss of generality, for clarity of presentation we assume a simplified binary sensing model of the two activities. Namely, an event of interest is positively detected if the drone is within a distance r_d (*detection range*) while event inspection requires a possibly shorter distance r_i (*inspection range*), with $r_i \leq r_d$. The drones may also exchange information with each other, and with their respective base stations located at the home depots. For simplicity of discussion, we also assume a binary communication model, with *communication range* r_c so that any two devices can communicate with each other only if their distance is lower than r_c.
In section 3.2.5 we provide an example of inspection and detection tasks using on-board cameras of the drones.

Environment and Probabilistic Map

We consider an AoI in which drones perform the monitoring tasks. The environment hosts temporally and geographically correlated events occurring dynamically during the mission, in the shape of hot-spots in the map, reflecting the most critical zones. The priori knowledge of the events location can be represented as an uncertainty map which is a function of the probability distribution of the target in the area. Areas that most likely host events will have higher probability, and vice versa. If no prior knowledge is available, the area is considered uniformly critical.

As the drones fly at approximately constant height to monitor the area, with a limited sensors footprint, we consider a 2D tessellation $\mathcal{A}$. Each cell $c \in \mathcal{A}$ is identified by its center coordinates (x_i, y_i), and has an edge of $c_{edge} = r_i \cdot \sqrt{2}$, where r_i is the drone inspection range, in such a way the drone can monitor the cell by hovering above that. We denote with $|\mathcal{A}|$ the number of cells in the AoI. This uncertainty map associates each cell c with the current estimate of the probability $p_c \in [0, 1]$ that cell c will host a critical event in the future of the mission.

Notice that, while this map is used to model uncertainty and the probability to host new events, the map can also reflect heterogeneous targets' relevance: important (or critical) targets can have higher relevance than others, attracting the drones and speeding up their inspection.

Problem Formulation

In this section we formulate the path planning problem under uncertainty.

We consider a set $\mathcal{U}$ of battery-powered drones, a set of depots $\mathcal{D} = \{d_i\}_{i \in N}$, and we represent the uncertainty map with a complete undirected connected graph $G = (V, E)$ s.t. $V \triangleq \mathcal{A}$. The mission lasts for T time (e.g., the mission is executed during daylight hours) or when all the targets has been visited. We hereby shortly denote with $[T] \triangleq \{0,\ 1,\ 2, \ldots,\ T\}$ the discretized mission instants. Each vertex $i \in V$ has an associated value $p_i \in [0, 1]$, which is the probability to host a critical event, while each edge (i, j) has an associated weight ω_{ij}, which represents the travel time for the drones.

First, we introduce a set of binary decision variables $x^u_{ij}(t) \in \{0, 1\}$, for (i, j) in V, $u \in \mathcal{U}$ and $t \in [T]$, to define the drone trajectories. The variable $x^u_{ij}(t)$ is 1, if the drone u_i leaves node i at time step t, to reach node j; it is equal to 0 otherwise. For simplicity we assume $\omega_{ij} \in [T]$.

We let drones start their mission at their base-station, at time $t = 0$, with the following constraint:

$$\sum_{i \in V} x^u_{d_u i}(0) = 1,\ \forall u \in \mathcal{U}.$$

where d_u is the home depot of drone u, where it is located at the beginning of the mission.

To avoid that the solution mistakenly provides multiple trajectories for a single drone, we include the following constraint

$$\sum_{i,j \in V \cup \mathcal{D}} x^u_{ij}(0) \leq 1, \forall u \in \mathcal{U}.$$

which requires drone u to fly over at most one edge at time 0.

Then, we constraint cyclic trajectories for all the drones with the following constraint:

$$\sum_{i \in V \cup \mathcal{D}:\ \omega_{ij} \leq t} x^u_{ij}(t - \omega_{ij}) = \sum_{i \in V \cup \mathcal{D}} x^u_{ji}(t), \quad \forall j \in V \cup \mathcal{D},\ t \in [1, \ldots, T], \forall u \in \mathcal{U}. \tag{3.16}$$

Finally, we require drones to return at any of the depots at the end of the mission, by means of the following constraint:

$$\sum_{i \in V \cup \mathcal{D}, d \in \mathcal{D}} x^u_{id}(T - \omega_{id}) = 1,\ \forall u \in \mathcal{U}.$$

To avoid possible position conflicts over the same targets, we introduce the following constraint:

$$\sum_{i \in V,\ d \in \mathcal{U}} x^u_{ij}(t) \leq 1, \forall t \in [T], j \in V \cup \mathcal{D}.$$

Notice that, two drones can instead come back (leave) to the depot together, as the landing (take-off) operations are executed sequentially by the drone itself.

For each target $i \in V$, we define the hovering time h_i, as the time required to collect data and complete the inspection of a possible event. Thus, we introduce a new variable to determine whether a target $i \in V$ is inspected or not. The binary variable $y_i^u(t) \in \{0,1\}$ is 1 if the drone u completes the inspection of i at time t, 0 otherwise. The variable y is constrained to the edge variables as follows:

$$y_i^u(t) \leq \sum_{z=max(0,t-h_i)}^{t} x_{ii}^u(z)/h_i, \ \forall i \in V, u \in \mathcal{U}, t \in [T]. \tag{3.17}$$

In agreement with our energy model, we consider battery-powered drones with a limited battery of b_u energy units. The drones consume α energy units for each time unit of flight, requiring battery recharging or replacement every $\delta_u \triangleq \left\lfloor \frac{b_u}{\alpha} \right\rfloor$ time units.
To represent the drone batter limitation, we define a new binary variable $k_d^u(t) \in \{0,1\}$ for each depot $d \in \mathcal{D}$. The binary variable $k_d^u(t)$ is 1, if the drone u completes the maintenance operations at depot d at time t, 0 otherwise.

$$k_d^u(t) \leq \sum_{z=max(0,t-\Delta)}^{t} x_{ii}^u(z)/\Delta, \ \forall d \in \mathcal{D}, u \in \mathcal{U}, t \in [T]. \tag{3.18}$$

where Δ is the average time needed for the maintenance operations (i.e., battery replacement). The variable $k_d^u(t)$ tracks how many time a drone u come back to the depot d for maintenance operations. Therefore, we add the following constraint to deal with limited drone lifetime:

$$\sum_{d \in \mathcal{D}} \sum_{z=0}^{\delta_u+\Delta} k_d^u(t+z) \geq 1, \ \forall t \in [T], \ u \in \mathcal{U}, \tag{3.19}$$

which requires that the maximum time between two consecutive maintenance operations is less than the drone lifetime δ_u, plus the maintenance time Δ.

Finally, we define the objective function of the problem, which maximizes the weighted sum of visited targets. We introduce an auxiliary variable $y_i \in \{0,1\}$ which is 1 if the target $i \in V$ is visited, and 0 otherwise:

$$y_i \leq \sum_{t \in [T], u \in \mathcal{U}} y_i^u(t), \ \forall i \in V. \tag{3.20}$$

We then define the following objective function:

$$max \sum_{i \in V} y_i \cdot p_i + \epsilon \cdot \sum_{i,j \in V \cup \mathcal{D}, u \in \mathcal{U}, t \in [T]} x_{ij}^u(t) \cdot \phi_{ij}. \tag{3.21}$$

The first term maximizes the weighted sum of visited targets, while the second term minimizes the flight time of the drones. To avoid that maintenance operations are included in the flight time, we consider $\phi_{ij} = 0$ if $i = j \wedge j \in \mathcal{D}$, otherwise $\phi_{ij} = \omega_{ij}$.

Theorem 3.2.1. *The Problem of Trajectory Planning under Uncertainty is NP-Hard.*

Proof. The addressed problem generalizes the Euclidean Traveling Salesman Problem, which is NP-Hard [127, 128]. Given a set of points $\mathcal{P} = p_1, ..., p_n$ in a metric space, the Euclidean Traveling Salesman Problem requires to find a tour τ that visits all the target points and minimizes the tour cost.

We can reduce any instance of the Euclidean TSP to an instance of our Trajectory Problem, by considering a single drone u, a set of known target points $V = \mathcal{P}$ with the same probability, i.e., $p_i = 1,\ \forall i \in V$, and a single base station $d \in V$. For each node $i \in V$ we consider zero hovering time ($h_i = 0$); for each edge (i, j) $\forall i, j \in \mathcal{E}$ we consider a cost ω_{ij} equal to the Euclidean distance between the two nodes, and we set the maximum flight time of the drone $\delta_u = \sum_{i,j \in V} \omega_{ij}$.

Thus, our problem formulation finds a tour τ for the drone u that maximizes the number of visited targets, with a minimum length, which is also the solution to the Euclidean TSP. The reduction is clearly polynomial, implying that our problem is at least as hard as the Euclidean TSP. □

3.2.3 SIDE

The proposed path planning problem under uncertainty is NP-hard, and intrinsically hard to solve. The optimal formulation is impracticably, especially in critical scenarios (e.g., in the aftermath of an earthquake), where drones have to be deployed immediately, to quantify the damages or help survivors. In fact, it would require hours of computation before the drones can depart from their depots.

To this end we propose SIDE, a novel distributed system for path planning under uncertainty. SIDE coordinates the drones to monitor the area of interest in searching for events while constantly improving the initial uncertainty map.

System Model

The overall monitoring system, presented in Figure 3.9, is composed of two main components: the depots and the drones. The depots cooperatively exchange the data collected by the drones and run the *Probabilistic Map Manager*. By means of this algorithm, the depot stations support the management of the mission execution. In particular, they iteratively update the uncertainty map, according to collected data.

The drones, which receive the probabilistic map from the depots, update their future trajectories accordingly, by means of our distributed path planning algorithm, called *SIDE*. The drone trajectories are computed during the mission, in a distributed manner, using a virtual-force approach, to handle local findings along the field and to allow drones to join/leave the mission at any moment.

Notice that depot stations support the construction of a global view of the probabilistic event map, by updating their view anytime a drone approaches them. In fact, we assume that the drones cannot perform all the updates on-board, due to their limited knowledge of the environment and events; and we point out that the movement protocol of the drones does not preserve connectivity, hence some

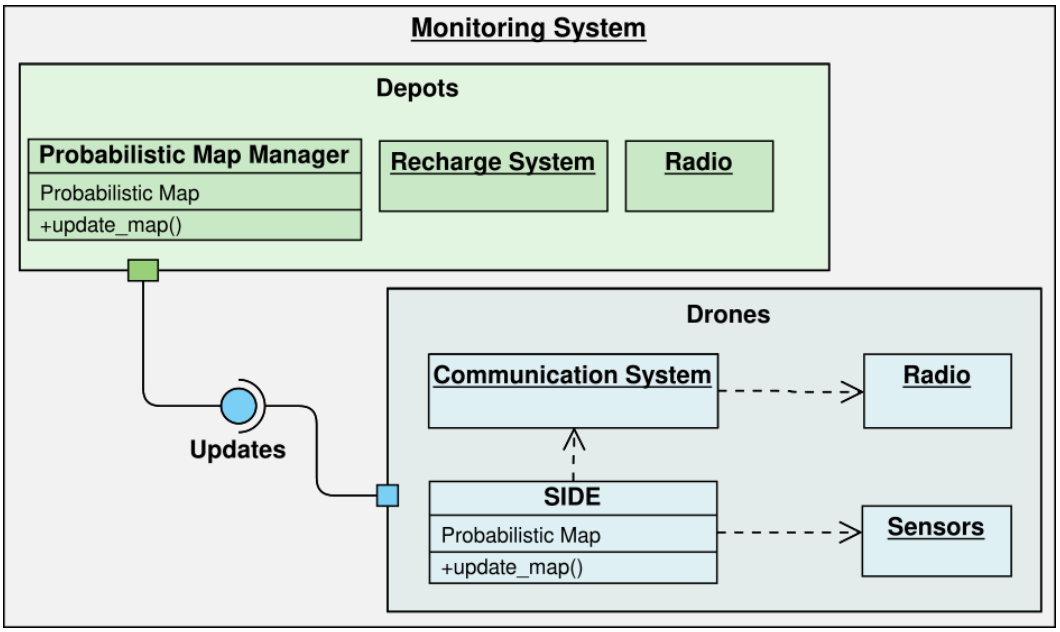


Figure 3.9. System Model

drones may have an insufficient and limited view of the environment. The drones only perform local updates, upon their local findings, and can eventually request help to other drones during the monitoring operations.

Probabilistic Map Manager

As we consider scenarios in which the events are geographically correlated, we represent the environment using a probabilistic map, whose global view is maintained at depot stations.

The map is continuously updated during the mission, according to a continuous multivariate probability density function (PDF), which describes how likely events of interest occur in each zone.

The PDF is built and updated using data from newly notified events, by means of the multivariate kernel density estimation (KDE) method, also known as the Parzen-Rosenblatt window method [121] [122]. KDE is a non-parametric way to estimate the probability density function of a random variable and it is a generalization of the histogram density estimation where the contribution of each event is smoothed into the region surrounding it. The smoothing area around an event captures the location dependent probability of new critical events occurring in its proximity. This method is often used to represent geographically correlated events which occur around hot-spots in the area or close to past events [129] [96].

In detail, KDE considers the observed event positions, smooths them with a kernel function and then it estimates the probability density function by summing up these kernels with a smoothing parameter $\boldsymbol{h} > 0$, called bandwidth. A kernel function $K(\boldsymbol{x})$ can be a continuous function such as Gaussian, Epanechnikov, Uniform, or Triangular. In our system we use a Gaussian Kernel with a unit variance but, a different kernel can be used to model any kind of smoothing around the phenomena location [130]. Thus, in our approach, each monitored event is smoothed using a

bi-variate normal distribution:

$$\mathbf{Y} \sim \mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.22}$$

where $\boldsymbol{\mu} \in \mathcal{R}^2$ is the event location, represented as a 2-dimensional mean vector, while $\boldsymbol{\Sigma} \in \mathcal{R}^{2\times 2}$ is the covariance matrix. This matrix reflects the smoothing area around an event, and consequently, the extent and shape of the risk area around it.

Finally, we estimate the probability density function of the map by using the multivariate kernel density estimator. Let us consider a set of N 2-dimensional input vectors, representing occurred events. The estimated probability density function is:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{N\boldsymbol{h}} \sum_{n=1}^{N} K\left(\frac{\boldsymbol{x} - \mathbf{Y}_n}{\boldsymbol{h}}\right). \tag{3.23}$$

The bandwidth $\boldsymbol{h}$ is computed by minimizing the mean integrated squared error [130] and affects the accuracy of the PDF construction, its smoothing and its orientation.

Figure 3.10 represents an example of the KDE estimation process starting from raw points, i.e., event locations (a), through kernel smoothing (b), up to the estimated density function (c). The third figure (c) shows the estimated critical zones from highest to lowest hazard (red to yellow, in the given heat-map).

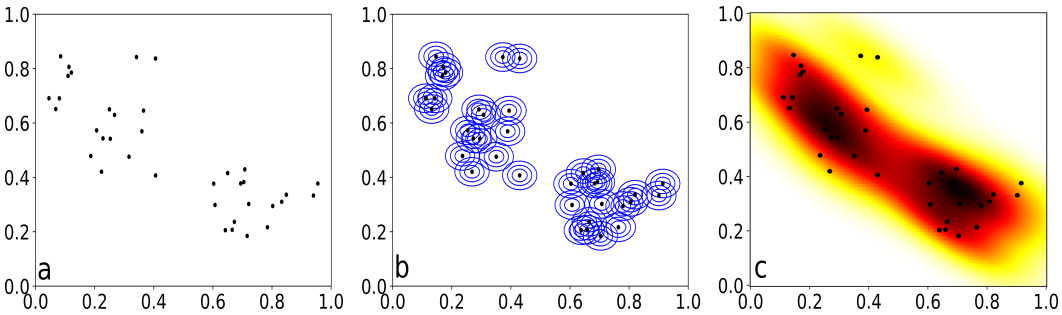


Figure 3.10. Examples of KDE

This probabilistic representation is an abstraction of the real environment phenomena. Thus, during the execution of the monitoring mission, as new data become available, the probabilistic map is updated and refined to produce a more accurate view of the ongoing events. At each update, the PDF is used to label cells in the partitioned area of interest $\mathcal{A}$ with their probability to host critical events. The probability p_c for a cell c is computed by integrating the PDF over the cell area, namely:

$$p_c = \int_c \hat{f}(\boldsymbol{x}) d\boldsymbol{x}, \qquad \forall c \in \mathcal{A} \tag{3.24}$$

To prevent neglection of some zones potentially occurring as a consequence of a biased estimation, we consider a lower bound of ϵ for the probability p_c when the value obtained by Equation 3.24 is lower than ϵ. The so constructed probabilistic map is then shared by the depot stations with all the drones at any contact, which will consequently update their local views.

3.2.4 Path Planning Algorithm

The path planning algorithm SIDE is a novel approach to trajectory management that conjugates the use of a classic virtual force approaches (VFAs) to swarm movements and the use of the dynamic event probability map.

Our virtual force model considers repulsive drone to drone forces to expand the network deployment and prevent collisions. It also incorporates attractive force sources exerted by event locations as determined by the probability map, to guide devices towards events and to avoid uninteresting or already visited spots.

Virtual forces in SIDE

SIDE is our distributed path planning algorithm. It is mainly based on a virtual force approach but it does not converge into a static deployment, instead, it aims at providing a full coverage of the area of interest with better surveillance of critical zones. According to SIDE each drone calculates the resultant of the composition of several force components to ensure event coverage and provide dynamic network deployment.

During the mission, each drone $u \in \mathcal{U}$ experiences attractive or repulsive forces determined by ongoing events, by the other drones and by the boundaries of the AoI which intersect its sensing and communication range. The resultant force $\vec{F}_u$, calculated by the vectorial sum of all individual forces, acts on the drone to determine the direction to travel during the mission. In the following we describe these forces.

Definition 3.2.1 (Drone to drone)**.** *The drone to drone force, $\vec{F}_{ui}$, is exerted by a drone i to repel a drone u depending on their mutual distance, to avoid overlapping of sensing areas and to increase the diffusion of the drones over the area. In particular:*

$$\vec{F}_{ui}(i) = \begin{cases} w_d \cdot \frac{1}{d(u,i)} & \text{if } d(u,i) \leq min\{r_d, r_c\} \\ 0 & \text{otherwise} \end{cases}$$

where $d(u,i)$ is the distance between the drones, and r_c and r_d are the communication and detection range of drone u, respectively. In particular, a repulsive force of w_d intensity is exerted when drones i and v are too close to each other.

Aerial collisions. Drone-to-drone force is mainly used to provide dynamic coverage and reduce redundant inspection but, it also avoids potential collisions between drones. In fact, a repulsive force is exerted when two drones are too close to each other. However, a drone that is returning to the base station, for battery replacement, maintenance, or data off-loading, does not exert, nor experience any force, to speed-up these operations. In that case, to avoid collisions the drones use different flight heights: while drones monitor the area flying at approximately constant height during the mission, they return to the depot using different flight heights. In case of shared or same depot, the take off and landing procedures are performed in a serial schedule.
We also recall that drones may directly integrate obstacle avoidance systems [54] or they can adopt online collision avoidance mechanisms [70].

In case of new event detection, an attractive force is exerted on the drones towards the event cells.

Definition 3.2.2 (Event to drone)**.** *The event to drone force, $\vec{F}_{ue}$, is exerted from detected, un-inspected events, to drone u. This force models the attraction of drones towards target cells that host detected events to be monitored. In particular:*

$$\vec{F}_{ue}(c) = w_e \cdot \frac{1}{d(u,c)}$$

where c is the cell which hosts the events, $d(u,c)$ is the distance between drone u and cell c, while w_e is the attractive intensity. Notice that, as soon as the events in the cell are inspected, this force component is set back to zero by the inspecting drone as well as by any drone who is notified of the occurred inspection.

This force is introduced to cause the inspection of newly detected events, i.e., to allow the nearby drones to include the detected event in their tasks. In particular, it is used to move drones towards newly detected events which need closer inspection (see Section 3.2.2).

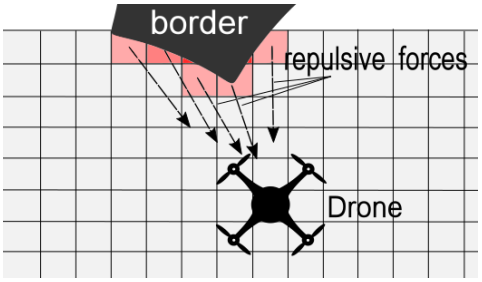


Figure 3.11. Virtual forces on borders

To prevent the drones from crossing the boundaries of the AoI during their flight we model a repulsive force exerted by the boundary, as shown in Figure 3.11, similarly to the approach proposed in [131].

Definition 3.2.3 (Border to drone)**.** *A border cell occupied by a drone u exerts a repulsive force $\vec{F}_{ub}$ on the drone itself, directed from the boundary of the cell c towards the drone position. In particular:*

$$\vec{F}_{ub}(c) = \begin{cases} w_b \cdot \frac{1}{d(u,c)} & \text{if } d(u,c) \leq r_d \wedge c \text{ is occupied,} \\ 0 & \text{otherwise} \end{cases}$$

where $d(u,c)$ is the distance between u and the center of the border cell c occupied by u, and w_b measures the repulsive force from the border cell, exerted in the direction of the interior of the AoI. The value of w_b is proportional to the area of the cell which falls out of the AoI (i.e. a cell mostly outside of mission area will have a high repulsive force).

Finally, as the algorithm aims at providing better surveillance of critical zones, we introduce the force exerted by the probabilistic map.

Definition 3.2.4 (AoI to drone)**.** *The AoI to drone force, $\vec{F}_{u\mathcal{A}}$, is exerted by cell $c \in \mathcal{A}$ according to its probability p_c to host critical events. This force guides the*

drones towards critical areas. In particular:

$$\vec{F}_{uA}(\mathcal{A}) = \begin{cases} \sum_{c \in r_d(\mathcal{A})} \frac{w_c}{d(u,c)}, & \text{if } r_d(\mathcal{A}) \neq \emptyset \\ \frac{w_c}{d(u,c)} \ s.t.\ c \in \eta(\mathcal{A}), & \text{if } \eta(\mathcal{A}) \neq \emptyset \\ 0, & \text{otherwise} \end{cases}$$

where: $r_d(\mathcal{A}) = \{c \mid c \in \mathcal{A} \wedge \ p_c > 0 \wedge \ d(u,c) \leq r_d\}$

$$\eta(\mathcal{A}) = \{c \mid c \in \mathcal{A} \wedge p_c > 0 \wedge \ \nexists c' \ s.t. \\ (p_{c'} > 0 \wedge \ d(u,c') < d(u,c)) \\ \vee (p_{c'} > p_c \wedge \ d(u,c') = d(u,c))\}$$

In words, $r_d(\mathcal{A})$ defines the set of all critical cells (i.e., $p_c > 0$) inside the drone detection range r_d, while $\eta(\mathcal{A})$ represents the most critical cells among the closest ones to the drone. This force uses the probability of neighbor cells (i.e. cells in the detection range of a drone) to host an event of interest, as defined by the probabilistic map. The purpose of this force is to move the drones towards the most critical cells. Therefore, the drone computes the overall force by using the cells in $r_d(\mathcal{A})$, if any, otherwise it picks one cell from $\eta(\mathcal{A})$ (a closest cell with non null probability value) to compute this force. If all the cells have null probability (i.e., $\eta(\mathcal{A}) = \emptyset$) the drones move back to the base station, to obtain an updated map or eventually end the mission. It is worth to mention that, this force is responsible to move the drones across the area of interest, by producing a sort of gradient ascent towards critical zones.

SIDE details

Algorithm 6 shows the procedure used by any drone u to monitor the area and determine its movements. As we consider battery powered drones, each drone continuously evaluates its residual energy, according the energy consumption model. It checks whether it is safe to carry on the mission, or its residual energy is just enough to move back to the closest base station for recharging operations (lines 4-6). In the algorithm, ω_u represents the energy consumption model, which maps flight distance to the required energy units, while ϕ_u denotes the current residual energy units of drone u. Then, the drone monitors the environment to collect data and detect events. It also updates the set of events called $\mathcal{E}$, by adding new detected events and removing inspected, or expired events (line 7). To share monitoring data and information concerning detected events, u communicates with neighbor drones and depots. Moreover, if its battery is about to deplete, u can request support from neighbor drones, by using special purpose messages (line 8). The drone updates its local map, using also received data, to avoid redundant coverage and update the information regarding critical zones (line 9). If the area, in the drone local view, has been covered entirely, the drone moves back to the closest base station to receive mission updates (i.e., a new map with critical zones to monitor) or concludes the mission. Otherwise, the drone computes the resultant force $\vec{F}_u$, by summing up all the force components described in Section 3.2.4, and moves accordingly (lines 11-19). Notice that, as the events are unknown, and they dynamically appear in the area,

Algorithm 6: SIDE

Input: Drone u, AoI $\mathcal{A}$, Depots $\mathcal{D}$, available energy ϕ_u, energy consumption models $\omega_u : \mathbb{R} \to \mathbb{R}$

```
events E, mission info I ← ∅
while (not mission completed) do
    neighbour drones U ← ∅
    b̂ = arg min_{j∈D}(d(u, j)) #closest base-station
    if φ_u − ω_u(d(u, b̂)) ≲ 0 then
        move to b̂
    collect_sensor_data(E, I)
    communication(U, D, E, I)
    update(A)
    if Σ_{c∈A} p_c = 0 then
        move to b̂
    F⃗_u ← F⃗_uA(A)
    for border cell c ∈ A do
        F⃗_u += F⃗_ub(c)
    for event cell c_e : e ∈ E do
        F⃗_u += F⃗_ue(c_e)
    for drone i ∈ U do
        F⃗_u += F⃗_ui(i)
    move_drone(F⃗_u)
    update(φ_u, ω_u)
```

the algorithm may go on indefinitely, provided that drone batteries are replaced or recharged upon need. The decision to quit the mission can be made by: A) the depot stations, when drones come back to recharge or get a new updated map, B) each drone, according to a timestamp which establishes the end of the mission after a predefined time (e.g., a mission may be executed only for some daily hours).

We also highlight that SIDE has the advantage to abstract and incorporate any energy model ω_u, which can be selected according the scenario and the drones. In particular, the algorithm can incorporate complex and non linear models, e.g., those proposed by Goss et al. [63]; or the consumption model reported by Sun et al. [132].

The communication protocol and the map updates are discussed in detail in the next two sections.

Communication Protocol

An essential aspect of SIDE is the communication between drones and depots, necessary to update the probabilistic map, and for the computation of the virtual force acting on each drone. In particular, as drones can be spread across the area, they usually have partial knowledge of the environment and of the mission. Only the base stations located at the depots, which are assumed to be connected and to receive updates from the drones, have global situation awareness.

According to SIDE, the drones periodically broadcast *hello* packets with the following data: the *drone ID*, the *drone position* (used for the force computation), the *version ID* of their map. In particular, the latter field is used to allow drones to recognize the need for a map update and perform it correctly.

Therefore, the drones use the following special purpose messages: 1) *map-update* which contains the probabilistic map, and is used to send updates or to notify visited zones and prevent redundant inspections; 2) *inspection-help* which contains an event coordinates and is used to attract neighbor drones towards an event that the sender drone cannot fully inspect due to limited energy availability (notice that, this message is answered with an ACK to stop the broadcast of help messages); 3) *event-updates* which contains the coordinates of an inspected event, to avoid redundant inspections. In particular, as map and event update messages may have non negligible size, they are only sent at periodic intervals, or in response to *hello* messages with older map *version IDs*, to force a map update.

Local and Global Probabilistic Map Updates

While the base stations manage global updates of the probabilistic map, to identify new critical zones, the drones perform some local updates upon collection of new data. These updates are used to guarantee system responsiveness and to provide an optimized coverage of the area. We now introduce the local and global update policies. These operations are of key importance for ensuring full coverage of the area of interest, especially in the absence of prior knowledge of event positions.

Local Update. During the monitoring operations, SIDE prevents unnecessary visit of already inspected zones (cells) by ensuring local map updates, and related information exchange with drones in radio proximity. For this reason SIDE provides a mechanism for updating information concerning the performed inspection operations. Whenever a drone inspects a cell c, i.e. the cell lies inside the detection range of a drone, the drone updates the related information in its local view of the map of events, by progressively decreasing its event value probability p_c during the monitoring time interval, according to a linearly decreasing function with finite *slope parameter* $\Delta \in (0, 1]$. Thus, the longer the time spent by a drone in cell c is, the lower the value of p_c will be. Notice that the particular setting of the parameter Δ impacts the amount of time that a drone spends on each zone (cell) of the area of interest, and in particular, defines the inspection time that will be devoted to critical zones.

It is worth observing that this map update operation is key to ensure complete coverage of the area, as we underline in the proof of Theorem 3.2.2.

Global Update. The global map update is managed by the base stations located at the depots and is performed according to the KDE method described in Section 3.2.3. We recall that, any time a global update is performed, the new probabilistic map is broadcast to the drones, and replaces the previous version of the map. Notice that, if two drones have different versions of the map, the old map is immediately updated. To avoid continuous replacements, that may cause starvation and lack of coverage in some areas (drones may cover only some zones), the base stations perform the global update only when every cell of the area has been explored at least once by at least one of the drones of the swarm. A regular

frequency of updates can be easily added in the model as a minor change, requiring the drones to regularly visit the depot. Moreover, when a drone depletes is battery and lands to the base station, it may eventually updates its uncertainty map.

Algorithm properties

While virtual force approaches do not guarantee coverage completeness (i.e. mobile devices can be trapped in local minima and fail to monitor the area), under specific conditions, and with a correct parametrization, SIDE covers all the area of interest in finite time. In the following we formalize these conditions.

Theorem 3.2.2 (Termination)**.** *Under the assumption that: (1) the battery availability of each drone u is sufficient to let u traverse the diameter of the AoI (i.e. the maximum distance between any two points in the AoI), (2) there are no obstacles in the aerial space, (3) no new event occurs, (4) batteries can be recharged or replaced at depot stations, SIDE eventually terminates in $O(|\mathcal{A}|)$ iterations and provides a full coverage of the AoI.*

Sketch. We consider a drone u and an AoI with a set $\mathcal{E}$ of unknown events at the mission start. We note that the dynamic generation of new events would prevent termination of the algorithm. However, new events are excluded by assumptions. By construction each cell $c \in \mathcal{A}$ has initial probability $p_c \geq \epsilon$. Let $f(\mathcal{A}) \to \mathbb{R}$ be the function defined as $\sum_{c \in \mathcal{A}} p_c$. Whenever a drone visit a cell c, the related probability value p_c is decreased of an amount equal to $min\{p_c, \Delta\}$, until it reaches zero and the inspection of cell c is terminated.

We note that under the given assumptions we cannot have unreachable cells. In fact, a cell may be unreachable because of energy limitations or area boundaries precluding drone movements. However, 1) the energy limitation can only affect the visit delays: we are assuming that any point is reachable by a drone, at least when its battery is full, and batteries can be recharged or replaced; 2) there are no obstacles, thus the drone can reach a cell starting from any position.

We also note that as long as $f(\mathcal{A}) > 0$, there is at least a cell c with non null probability $p_c > 0$. Hence, if $f(\mathcal{A}) > 0$ SIDE will continue moving the swarm until all cells are inspected and eventually reach zero probability as a consequence of the local update mechanism. This is because either the drones are inspecting some cells, whose probability will eventually reach zero, or they are moving towards other cells with non null probability, or they are moving to the depot from which they will receive global updates concerning the presence of unvisited cells, if any, to which they will move accordingly to ensure further inspection and eventually complete coverage.

We also note that the local map update mechanism also ensures the algorithm termination. In fact, we note that the a necessary and sufficient condition for termination is that $f(\mathcal{A}) = 0$ which implies that the swarm can stop as, by hypothesis, no new events will be generated to increase the value of $f(\mathcal{A})$. However, we observe that the value of $f(\mathcal{A})$ reaches zero in at most $|\mathcal{A}|/\Delta$ iterations, which lets us conclude that the algorithm terminates in at most the same number of steps. □

Notice that if events continuously appear in the AoI the protocol may incur starvation. To prevent this, the algorithm can use an appropriate scheduling policy

based on an aging algorithm, for instance the Earliest Deadline First (EDF) algorithm proposed in [133].

We now analyze the time and message complexity of SIDE. As the algorithm works online, as soon as new events appear in the area, the drone trajectories change and the algorithm does not stop. In the following we analyze the algorithm complexity with a given number of events $|\mathcal{E}|$, and we assume that the number of events $|\mathcal{E}| = \mathcal{O}(|\mathcal{A}|)$, where $|\mathcal{A}|$ is the number of cells in the AoI. While our protocol sends and asks updates at fixed intervals of time, our analysis considers a worst case scenario in which only one message is sent at each iteration.

Theorem 3.2.3 (Message Complexity)**.** *For each drone, the message complexity of Algorithm 6 is* $\mathcal{O}(|\mathcal{A}|)$ *for transmissions and* $\mathcal{O}(|\mathcal{A}| \cdot (|\mathcal{U}| + |\mathcal{D}|))$ *for receptions.*

Proof. The worst case message complexity can be calculated by considering a drone $\hat{u}$ with the maximum possible number of neighbors, i.e., $|\mathcal{U}| - 1$ drones and $|\mathcal{D}|$ depots. At each iteration, a drone broadcasts the message with its position, the map summary and the inspected or expired events. It also receives at most $(|\mathcal{U}| - 1)$ messages from neighbor drones, and $|\mathcal{D}|$ messages from depots, at each iteration. As the number of iterations is $O(|\mathcal{A}|)$, as stated by Theorem 3.2.2, the message complexity is $\mathcal{O}(|\mathcal{A}| \cdot (|\mathcal{U}| + |\mathcal{D}|))$. □

Theorem 3.2.4 (Time Complexity)**.** *The time complexity of Algorithm 6 is* $\mathcal{O}((|\mathcal{U}| + |\mathcal{D}|) \cdot |\mathcal{A}|^2)$.

Proof. The worst case time complexity can be calculated by considering a drone $\hat{u}$ with the maximum possible number of neighbors, $|\mathcal{U}| - 1$ drones, and $|\mathcal{D}|$ depots. Depending on the available residual energy, the drone may perform one of the following actions:

1. The drone has insufficient energy to continue the mission and interacts with at most $|\mathcal{D}|$ depots to determine the next recharging station.

2. The drone calculates the resultant of the forces exerted by other drones or cells of the map, which has complexity of at most $\mathcal{O}(|\mathcal{U}| + |\mathcal{A}|)$. It also updates the map with the collected and received data, which has complexity of $\mathcal{O}(|\mathcal{A}|)$ and $(\mathcal{O}((|\mathcal{U}| + |\mathcal{D}|) \cdot |\mathcal{A}|))$, respectively.

Therefore, at each iteration the time complexity is $\mathcal{O}((|\mathcal{U}| + |\mathcal{D}|) \cdot |\mathcal{A}|)$. As SIDE visits the area in $O(|\mathcal{A}|)$ iterations, according to Theorem 3.2.2, the time complexity is $\mathcal{O}((|\mathcal{U}| + |\mathcal{D}|) \cdot |\mathcal{A}|^2)$. □

3.2.5 Performance Evaluation

To the best of our knowledge, no previous proposal addresses the distributed trajectory design for a UAV squad in uncertain environments as we do in this work. Therefore, to enable fair performance comparisons we modified two existing works to make them suitable to our problem setting. The first algorithm, called *AC-GaP* (see Section 2.1), is designed to let a UAV squad visit a set of known points of interest with minimum average inspection latency. We recall that, given a set of

feasible trajectories traversing different subsets of the targets, the algorithm selects and schedules some of them into a multiple trip sequence, with recharging and off-loading operations between consecutive trips. In order to make AC-GaP suitable to our problem setting, we make it work with a problem instance where a number of targets is uniformly distributed over the area, with sufficient density, such that complete inspection of the targets implies complete monitoring of the area of interest. AC-GaP is then executed iteratively, with periodic optimization of the trajectory schedules which prioritizes the trajectories on the basis of the number of detected events, to obtain maximum inspection rate and minimum average inspection latency. The second algorithm, called *Sweep* is also a variant of a previous solution [53] modified to fit our problem setting. The algorithm was originally designed to guide the movements of multiple ground robot to ensure complete coverage of an area of interest. We modified the original algorithm by partitioning the area of interest into regions that can be explored completely within the energy constraints imposed by the UAV battery of a single drone. Each drone periodically scans each partition by using back-and-forth trajectories until the end of the mission.

Metrics

We consider missions with uncertain information, in which targets generation times and positions are unknown. Thus, evaluating an algorithm's effectiveness (i.e., percentage of monitored targets) is rarely possible, as the events are unknown.
In our simulations and real-field experiments to comprehensively evaluate the algorithms, we use an oracle that knows each target's exact position and generation time. The oracle is used to evaluate the performance of the algorithms, i.e., to compare the events collected by algorithms with respect to the events generated in the AoI.

We evaluate the following metrics. The *detection delay* measures the time between the generation of an event and its detection. However, the time between an event detection and its final inspection may be highly variable, due to energy or trajectory constraints. When a drone detects an event it may not have sufficient energy to inspect it. For this reason the event may be inspected by another drone or by the same drone in a subsequent trip, after a battery replacement. Thus, we also study the average *inspection delay*, defined as the average time between the generation of an event and its actual inspection by a drone (including hovering). Detection and inspection delays measure the algorithm responsiveness when events are successfully detected and inspected, respectively. Note that some events may remain undetected or not inspected because they disappear before the drone intervention. Therefore, we also measure the *coverage percentage*, defined as the ratio between the number of inspected and occurred events; and the *percentage of area covered* defined as the ratio between the explored area and total area to inspect.
Finally, we also evaluate the *computational time* defined as the time needed to compute the trajectories, over a laptop with a Intel i7-8665U CPU with 32gb of RAM.

Simulation results

Where not otherwise stated, we consider a 4-hour long mission, an area of 10 km^2, and 2 base stations located at the bottom corners of the area of interest. The inspection, detection, and communication radii are 150, 200 and 300 meters, respectively; the maximum speed of a drone is 10 m/s while its acceleration is $\pm 3.5 m/s^2$. Events occur every 120 seconds in random positions defined according to a bi-variate normal distribution centered around related epicenters. Events last for 30 minutes and require a hovering time of 30 seconds for complete inspection and collection of enough data. We assume that each event has a unique fingerprint, which allows us to distinguish between the same event or a subsequent event. Notice that, the fingerprint is mission-specific: a geyser eruption can use the GPS coordinates and the current time as fingerprint, as any eruption at different coordinates or time would be another independent event.
The drones running SIDE exchange *hello* messages every 0.5 seconds, while periodic *map-update* and *event-updates* are sent every 10 seconds.

While SIDE can incorporate any energy consumption model in the simulations we adopt a simplified model which translates the energy availability into residual flight time. According to this model a fully charged battery corresponds to 30 minutes of flight time, potentially including hovering [134].

To simulate the propagation channel we adopt a two ray propagation model [106], which suits our scenario with no obstacles between drones, and air-to-air (AA) communications. Notice that, the two ray propagation model is an extension of the free space model, which is commonly used in AA communications [104], but it also considers ground reflection.

The model computes the received power at distance d as follows:

$$P_r(d) = \frac{P_t \cdot G_t \cdot G_r \cdot {h_t}^2 \cdot {h_r}^2}{d^4 \cdot L} \tag{3.25}$$

where h_t and h_r are the heights of transmitter and receiver, respectively; G_t and G_r are their antenna gains; P_t is the transmitted signal power, and L is the system loss.

Where not otherwise stated, plots present the result of 40 simulation runs, and the error bars denote one standard deviation from the mean.

Optimal Trajectories In the first set of experiments, we compare the optimal path-planning defined in Section 3.2.2 with SIDE, when the size of the mission area increases. Given the significant computational requirements of the optimal formulation, we focus on small scenarios. We consider a mission of 5 minutes, with one base station and 2 drones. The drones fly at $5m/s$ and have radii of 200, 100, and 75 meters, for communication, detection and inspection. For the computation of the optimal solution, we consider an oracle that provides the uncertainty map, which denotes the areas where events appear. We have 9 events in the area, and we execute 5 runs for each experiment.

Figure 3.12 shows the percentage of area inspected by the two algorithms. SIDE's performance is close to the optimal solution, achieving around 90% of its coverage. It is interesting to note that the percentage of inspected area drops by increasing the

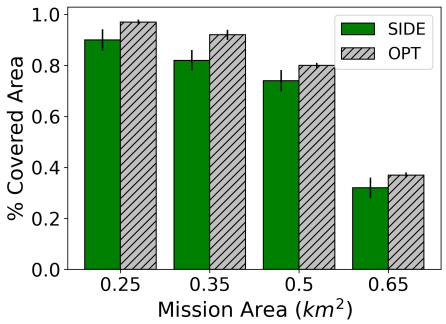


Figure 3.12. Percentage of Area Covered.

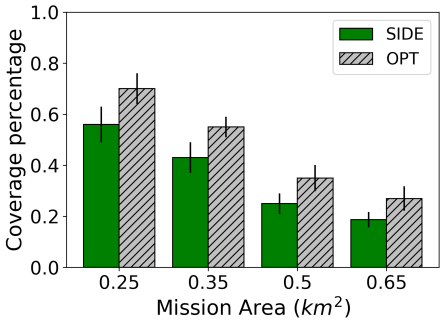


Figure 3.13. Coverage Percentage.

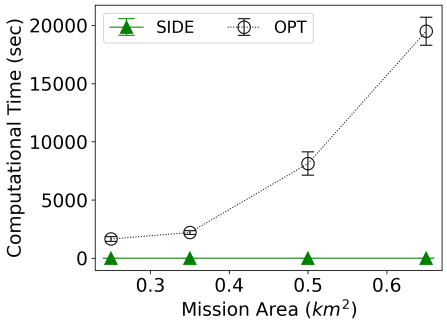


Figure 3.14. Required Computational Time for the solution.

area, as the number of drones is not sufficient to cover the entire area. In Figure 3.13 we evaluate the percentage of inspected targets. While the optimal solution always shows the best results, SIDE achieves 80% of its performance. The large performance gap between SIDE and the optimal solution is motivated by the better trajectories of the optimal approach, which guides the drones towards the zones hosting most of the events. Finally, Figure 3.14 shows the computational time of the two approaches. While for SIDE the computation time is almost zero, the computation time of the optimal grows significantly, due to the complexity of AoI tessellation. Therefore, the optimal solution is not applicable in safety-critical settings, however small.

Varying distribution of events around one epicenter We now investigate the performance of SIDE in comparison to two state-of-art approaches. To consider experiments in larger scenarios, we will not show the performance of the optimal solution, due to its computational complexity. In this setting we consider events occurring around a unique epicenter located in the center of the area of interest. We investigate several scenarios with increasing size of the impacted area, from scenario 1 to 6, whose event distribution is described in the heat-maps of Figure 3.15: out of the 120 events occurring during an observation period of 4 hours, we observe 40 events taking place in the border area, in the color range from green to white, while we observe 75 events in the range from blue to white.

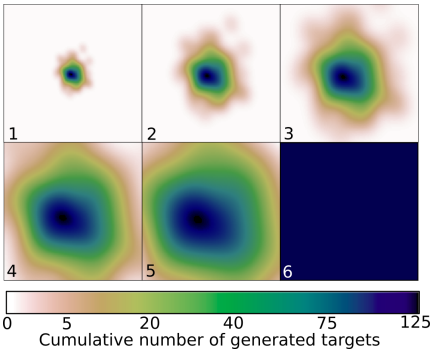


Figure 3.15. Points generation scenario

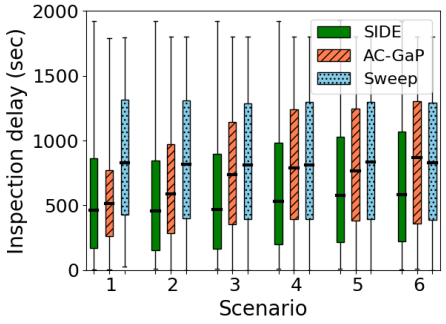


Figure 3.16. Inspection delay

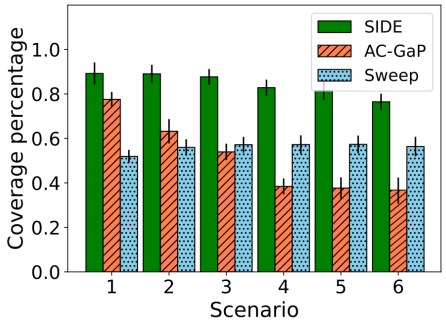


Figure 3.17. Coverage percentage

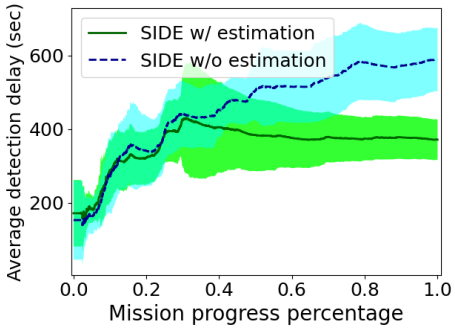


Figure 3.18. Detection delay (Scenario 2)

In the last scenario (i.e., 6) all events are instead uniformly distributed in the entire area.

We simulate a squad of 5 drones exploring the area. Figure 3.16 shows the inspection delay of the three algorithms. We use box-with-whiskers plots to highlight the quartiles of related events falling within each metric interval. The figure shows that SIDE is remarkably faster than the other algorithms in inspecting events, with an improvement that ranges between 27% and 40% with respect to the other solutions. This is due to the capability of SIDE to perform focused trajectories aiming at providing the required inspection for each upcoming event. This is confirmed in all the considered scenarios. The Sweep algorithm instead aims at providing complete area coverage, resulting in a performance that is not affected by the particular size of the affected area, and is almost uniform across the six scenarios. It is worth noting that although AC-GaP aims at prioritizing areas where events have been previously detected, it is not capable of taking into account the duration of the events, and has a high event miss rate. Indeed, it often visits the event locations too early or too late with respect to the actual period of event persistence. This is also confirmed in Figure 3.17 which evaluates the event coverage percentage. In the first three scenarios, SIDE is able to inspect over 90% of the occurring events. Its coverage is also the highest, and consistently above 80% even in the other scenarios, in which Sweep hardly achieves 60% and the performance of AC-GaP gradually drops below 40%. As the impacted area grows within the considered scenarios, the event distribution becomes more uniform, which makes Sweep perform better than

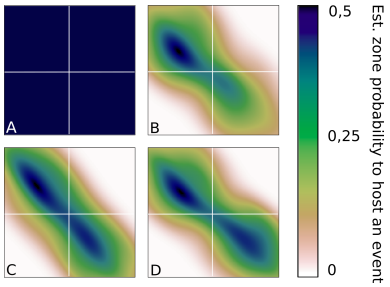


Figure 3.19. Progressive map estimation

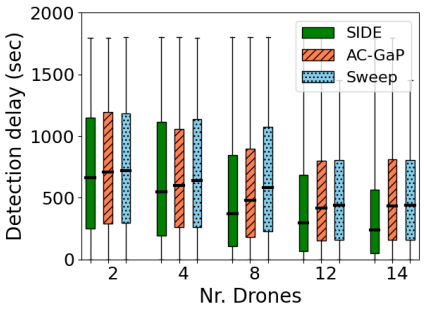


Figure 3.20. Detection delay

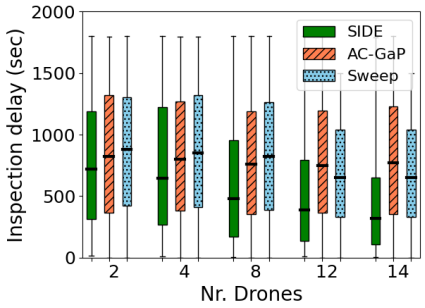


Figure 3.21. Inspection delay

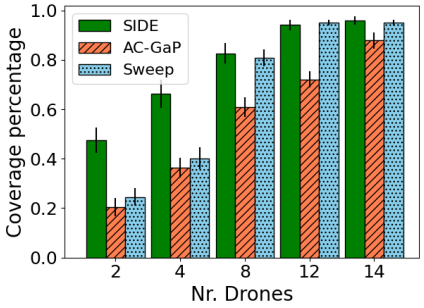


Figure 3.22. Coverage percentage

AC-GaP. Nevertheless, SIDE outperforms Sweep even in its most favorable scenario, i.e. the sixth.

Figure 3.18 evidences the powerful contribution of the map estimation of SIDE by comparing the average detection delay that is obtained with or without the map construction. The figure refers to the second scenario and shows the trend of the average detection time during the mission. The figure highlights how, after an initial training phase, SIDE with map estimation is able to reduce the detection delay of around 35% with respect to the same algorithm, without map construction.

Events occurring around two epicenters In our third set of experiments, we partition the area into four quadrants and generate the events of interest around two epicenters located at the top-right and bottom-left quadrant of the area of interest. In the heat-map of Figure 3.19 we show the progressive computation of the event probability map, during a simulated mission with 8 drones. The colors of the heat-map reflect the estimated probability to host a critical event during the mission. The map initially considers uniform event probability, shown in Figure 3.19-A, as drones have no initial knowledge of the events. Drones gradually refine the probabilistic map with the incrementally available information, as shown in Figures 3.19-B, 3.19-C, and 3.19-D, corresponding to 10%, 30% and 50% of the overall mission time (4 hours), respectively.

Figures 3.20 to 3.22 show the results by varying the number of drones. Figure 3.20 shows that the three algorithms need almost the same time to detect events when they work with only 2 drones. However, SIDE becomes remarkably faster than the other algorithms when the number of drones increases: with 14 drones the median detection delay of SIDE is half of the other algorithms' delay. This occurs because SIDE performs a focused search for upcoming events, while AC-GaP

Figure 3.23. BaseStation and DJI F550 Hex-rotor with Naza-M V2

is unable to prioritize event inspection before their expiration, and Sweep aims at complete area monitoring regardless of the event positions. Such a difference is even more remarkable when we consider the inspection time, shown in Fig. 3.21, where SIDE is always faster than the other algorithms. In the case of 12 and 14 drones, SIDE is able to inspect 50% of the events in the first 370s, resulting 76% faster than Sweep, which requires 650s. This improvement derives from the ability of SIDE to promptly respond to detected event by directly inspecting it or asking nearby drones for intervention.

SIDE outperforms the other algorithms also in terms of event coverage percentage. As shown in Figure 3.22, with only 2 drones, SIDE inspects 50% of occurred events, while AC-GaP and Sweep hardly reach 25% of event coverage. The percentage of inspected events increases to 75% when SIDE employs 4 drones, while it remains below 40% for AC-GaP and Sweep. For larger fleets (i.e. 12 and 14 drones) the coverage performance of all the algorithms improves significantly, with SIDE and Sweep hitting the 95%, and AC-GaP performing 33% and 9% worse than the other two algorithms. Nevertheless, in our experiments we see that the higher inspection delay of AC-GaP and Sweep is likely to have an impact on their coverage performance when the event duration decreases, while SIDE is more flexible to variation of this parameter.

Real-field experiments

We now experimentally investigate the applicability of our proposal in a real scenario. We emulate a real mission with several events that appear dynamically in the area.

The equipment we use for the test-bed includes a fleet of 4 programmable drones, equipped with cameras and GPS modules, and a laptop, needed to run the algorithms and compute the trajectories provided to the drones. Figure 3.23 shows one of the drones and a laptop that we used as a base-station. We use hex-rotor DJI F550 [55]

Figure 3.24. Detection of a target.

Figure 3.25. Inspection of a target.

drones equipped with LIPO batteries, 3500 mAh, 11.1 v, 25 C and a Raspberry Pi 2 with a WiFi Antenna [135]. Therefore drones can communicate among each others and with the base station (i.e., laptop). We consider a communication range of approximately 150 meters according to preliminary in-field measurements. With this setting, drones have enough energy to perform approximately 7 minutes of consecutive flight with a speed up to 5m/s. The field of view of the cameras is 10 meters (detection range) while we required a distance of less than 5 meters for proper inspection (inspection range). The experiments have been performed in a open field, and drones were deployed in an area of 140x90 meters.

In the experiments we consider a simplified model of the events. We emulate events of interest by deploying big red balloons along the field (see Figure 3.24). We distinguish between target detection and inspection on the basis of the distance between the target and the camera.
Figure 3.24 shows a drone image in which the drone detects the red balloon (i.e., the red ball with yellow contour), but the ball is not close enough to the image center (i.e., it is not inside the center yellow box), and therefore only partial data can be collected (i.e., detection).
Instead, in Figure 3.25 the drone centered the balloon under the camera, allowing to collect more precise data (i.e., inspection).
The targets are dynamically positioned by a human operator during the mission, to emulate new events. Notice that this setup models a scenario in which some sensors (e.g., microphone) can first detect a far new event (e.g., car accident) to later move and inspect it with a camera.

We initially validated our simulation models by studying similar settings with both simulations and test-bed experiments. This comparison revealed a difference of no more than 6% in terms of detection and inspection delays, which may be ascribed to environmental factors, mostly the presence of wind.

We instead show the results of 2 different sets of experiments, each 30 minutes long, in which we replace the drone batteries upon need.

The first one (Scenario 1) considers a series of 10 consecutive events starting 2 minutes after the squad take-off. The series of events begins in a location close to the top left corner of the area of interest. Each 120 seconds a new consecutive event appears in a range of around 15 meters from the first event location and it lasts

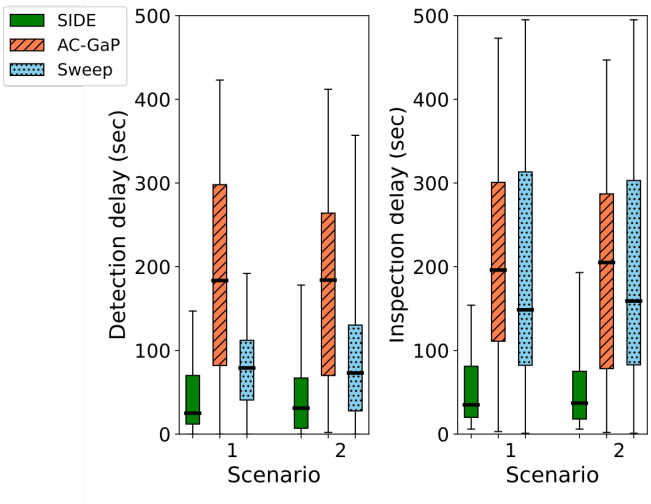


Figure 3.26. Real Experiments Results

about 100-150 seconds. This experiment models a collapsed building with survivors to be located and helped.

The second experiment (Scenario 2) considers a similar scenario, with the same number of events, appearing in a wider range from the first of the series, i.e. in random locations within a range of around 25-30 meters from the first event. It models a large size accident, such as a wildfire.

The experimental results, each based on 3 different runs, confirm the algorithm performance and trends. Figure 3.26 shows that although all the algorithms cover all the target points, SIDE significantly outperforms Sweep and AC-GaP both in detection and inspection delay. This is due to the incapability of AC-GaP and Sweep to focus on critical events if they are not known in advance, i.e., prior to the squad take-off. As a consequence, both these algorithms spend part of their mission time searching for upcoming events, consequently performing the related inspection with significant delay, if they do not miss them altogether, which can occur when the drone flies above the event location before or too long after the event occurrence.

3.2.6 Conclusions

We introduce SIDE, a novel distributed trajectory planning algorithm for multiple drones which autonomously coordinate in the exploration of a dynamic environment to monitor a set of geographically correlated events of interest. With no initial knowledge, the drones share their observation of the environment and gradually build a probabilistic map of ongoing events, which they update on the basis of their findings. SIDE uses this map and leverages virtual force methods to bring drones towards the most critical zones, avoiding collisions and monitoring redundancy. This is a new approach that is particularly useful for dynamic environments, in which a set of temporary and geographically correlated events may occur. Results show that SIDE outperforms baseline and state of the art approaches. In most of the considered settings SIDE halves the inspection delay and doubles the event inspection rate, with respect to previous solutions.

Chapter 4

Communication Protocols

In the previous Chapter 3 we exploited the communication capabilities of the drones to address safety critical missions. We discussed how drones can use multi-hop communication to share control and data packets, and autonomously coordinate during the whole mission.

However, the fast and unconstrained mobility of FANETs requires new solutions to enable stable and reliable communication between nodes. Previous work in this direction aims at extending protocols designed for Mobile Ad-hoc NETworks (MANETs) or Vehicular Ad hoc Networks (VANETs), but the proposed solutions do not fully address the unique characteristics of the UAV networks [7].

In this chapter we develop a new protocol for packet routing in the challenging domain of FANETs, considering an highly dynamic topology scenario. Unlike previous approaches, we exploit the device controllable mobility to facilitate network routing. In Section 4.1 we propose *MAD (Movement Assisted Delivery)*: a packet routing protocol specifically tailored for networks of aerial vehicles. MAD enables adaptive selection of the most suitable relay nodes for packet delivery, resorting to movement-assisted delivery upon need, which is supported by a reinforcement learning approach. By means of extensive simulations we show that MAD outperforms previous solutions in all the considered performance metrics including average packet delay, delivery ratio, and communication overhead, at the expense of a moderate loss in average device availability.

This chapter has been extracted from the work in [24].

4.1 MAD for FANETs: Movement Assisted Delivery for Flying Ad-hoc Networks

Multi-hop routing over FANETs is a considerable challenge and an open problem, due to the highly dynamic topology of these networks, to the fast variability of application demand, and to the inhomogeneous link quality of typical outdoor environments [7].

To date, the de facto standard for routing over FANETs has not been determined yet [12]. The majority of existing approaches consider variants of widely acknowledged protocols, originally designed for Mobile Ad-hoc Networks (MANETs) [136] and Vehicle Ad-hoc Networks (VANETs) [137]. Nevertheless, MANETs and VANETs differ from FANETs in terms of device mobility, topology variability, device speed, energy constraints, and also application features, the latter having considerable consequences on the typical traffic profiles. Previous solutions specifically designed for FANETs, fall short of taking complete advantage of the unique potentialities of these networks, in particular of the controlled mobility. While most of them rely on geographic approaches to find the most suitable relay nodes for communication [138] and device trajectories are only determined on the basis of the application tasks, other works employ dedicated data-ferries [139] to facilitate communications. These solutions result in high delivery delay and poor device utilization, which translates in poor responsiveness to local findings. Instead, we introduce a completely novel approach, called *Movement Assisted Delivery* (MAD), where flying devices actively learn the most suitable trade-off between the requirements of the application and communication tasks, namely maximum monitoring availability and minimum communication latency. According to our solution, whenever a UAV detects an ongoing critical event, it sends a message to the central processing center, located at the sink. For this purpose, it looks for an existing nearby relay node (Figure 4.1-a) or actively moves towards the packet destination (Figure 4.1-b), until it finds a suitable relay. After packet delivery, the UAV returns to its application task, to ensure continuous application availability (Figure 4.1-c). To the best of our knowledge, this is the first work in the literature so far to jointly address communication and application requirements in a unique solution for UAV trajectory design.

MAD adopts a Reinforcement Learning (RL) approach to guide device movements based on delivery needs (e.g., in case of neighbor device scarcity, or after a number of unsuccessful transmission attempts) and enables adaptive selection of the most suitable relay nodes, based on the position of the packet destination, nodes' speed and packet loss probability.

We analyze MAD in terms of delivery guarantees and time complexity. By means of extensive simulations, we give evidence of the beneficial impact of controlled mobility. We do so by evaluating MAD against baseline approaches with disabled mobility control. Then, we compare MAD to two existing solutions for packet routing in networks of aerial vehicles: QMR [138] and DTN [139]. We note that some protocols specifically designed for MANETS could be extended to consider UAVs' communications. For this purpose we conclude our experimental analysis comparing MAD with two MANET protocols: BATMAN [94] and GFG [140]. The experiments show that MAD outperforms previous solutions in all the considered performance

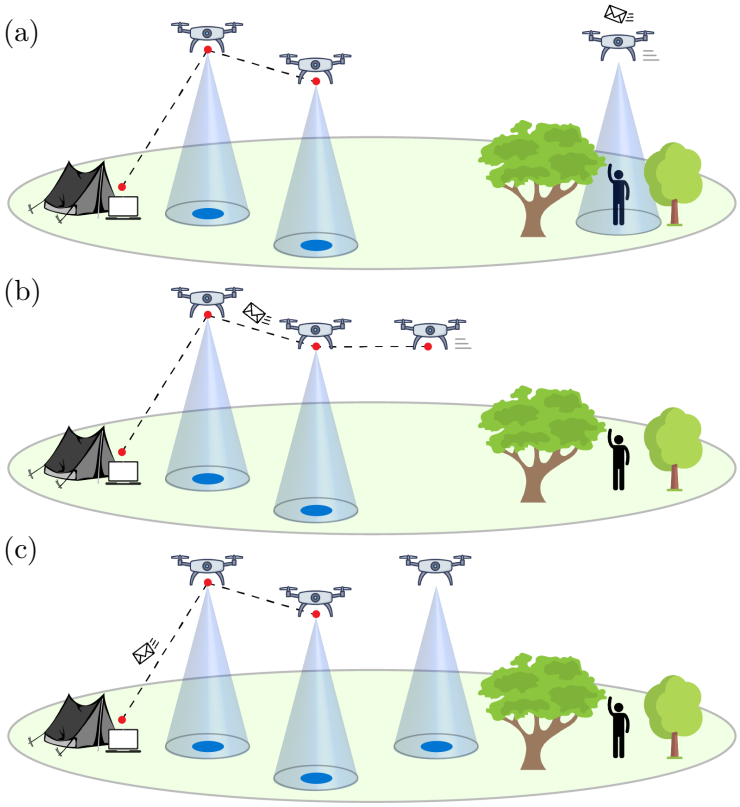


Figure 4.1. Event detection and communication using MAD

metrics including average packet delay, delivery ratio, and communication overhead, at the expense of a moderate, controllable loss in average device availability, due to its movement-based delivery.
The major contributions of this work are the following:

- We introduce MAD, a routing protocol for FANETs, that utilizes a RL approach to make adaptive decisions to facilitate packet delivery through controlled device movements.
- We define an adaptive relay selection scheme which considers measurements of data link quality, and an estimate of the node positions during the future radio transmissions.
- We formally analyze the fundamental properties of MAD and characterize its delivery guarantees and complexity.
- Through simulations, we first motivate mobility assisted delivery by comparing MAD with benchmark solutions with no mobility control. Then we show the superiority of MAD over DTN [139], QMR [138], BATMAN [94], and GFG [140], in all the considered performance metrics, at the expense of a moderate and controllable loss in device availability, due to the introduction of mobility assisted delivery.

4.1.1 Related Work

Most of current approaches consider variations of widely known protocols, originally designed for Mobile Ad-hoc Networks (MANETs) [12], which are classified in *proactive*, *reactive*, and *hybrid* protocols, depending on the route discovery strategy. A major pitfall of *proactive approaches* is poor responsiveness to the fast topology changes that characterize UAV networks. Optimized Link State Routing (OLSR) [141] is a routing protocol that periodically broadcasts link-state costs to allow computation and update minimum cost routes. BABEL [142] is also a proactive protocol which instead utilizes a distance-vector approach. BATMAN (Better Approach to Mobile Ad Hoc Network) [94] was specifically designed to deal with fast topology changes. The study in [7] demonstrates the superiority of BATMAN over OLSR and BABEL when applied to a FANET setting. For this reason we included BATMAN among our benchmarks for performance analysis. *Reactive protocols*, such as Dynamic Source Routing (DSR) [143] and the Ad hoc On-Demand Distance Vector (AODV) [144], compute routes only when needed. In FANETs this approach results prone to the problem of obsolescence of the computed routes, due to high devices mobility.

Geographic approaches are commonly considered the most promising protocols for FANETs thanks to their moderate network overhead [139, 140, 145–147]. In geographic routing, a UAV selects a relay so as to minimize the distance between the next hop and the destination of a packet, but possibly incurring in a local minimum, which requires protocol-specific countermeasures. AeroRP [145] routes packets closer to the destination based on a speed-based heuristic that is calculated for each one-hop neighbor. However, it is designed for high-speed (1200 m/s) aerial vehicles and requires full trajectory knowledge. Closest to our work are the DTN_{geo}, DTN_{close}, and DTN_{load} algorithms [139] that implement a motion-driven packet forwarding algorithm, applying delay-tolerant networking (DTN) in case of disconnection. These protocols exploit sensor information of UAVs to make location-aware packet forwarding using physical motion of UAVs. When a UAV is disconnected from the network, its data packets are carried by ferry UAVs. The protocol exploits long-range low-throughput communication to optimize flight behavior. As DTN_{close} addresses mission critical scenarios we include it as a benchmarks for performance analysis. However, we make no assumption on out-of-band channels, data ferries, or full trajectory knowledge.

In the context of RL based protocols, QGeo [148] and QMR [138] dynamically learn and adapt the packet routes to the application scenarios. QMR protocol employs a RL agent for each packet to select the next hop to destination, based on the residual energy available at the destination node and current packet delay. As QMR has shown to perform better than QGeo we include it in our performance comparison.

Unlike previous approaches, ours is the first that exploits device mobility of the whole fleet to facilitate packet delivery, without relaying on trajectory knowledge, dedicated ferry UAVs, or out-of-band channel. Our solution guides device movements and relay selection decisions on the basis of a geographic approach, supported by a reinforcement learning framework, trading off between application availability and delivery guarantees.

4.1.2 MAD Protocol

We consider a fleet of UAVs, composing the set U, hereafter referred to as *nodes*. The nodes are deployed in an area of interest with the goal of collecting sensing data. Upon detection of anomalies or critical events in the monitored environment, the nodes may need to communicate with the sink with maximum urgency. For this reason, in addition to data offloading at the end of the mission, we assume that nodes may be required to send critical packets to the sink to trigger immediate intervention. Nodes are equipped with a localization module (e.g., GPS), and separate transceivers for simultaneous transmission and reception activities. They move freely across the area of interest, making the network topology highly dynamic. Without loss of generality we assume that the nodes are aware of their current geographical destination, i.e., of their next way-point. Each node moves towards it at approximately constant speed following a linear trajectory. When it reaches its destination, a node selects a new way-point according to the application needs. This process repeats until the mission ends or the node exhausts its battery power.

Protocol Overview

We propose a new routing protocol called MAD (*Movement Assisted Delivery*), addressing node-to-sink communications, which combines adaptive selection of the most suitable relay node and movement-based delivery. Whenever a node has data to send (through a *data packet*, *dpk*) it looks for a suitable relay, i.e., a node that is, and is likely to be in the near future, in a better position for ensuring delivery to the sink. If a relay is found, the source node forwards the data packet. Otherwise, the node employs a Q-learning based strategy to decide whether to *stay* on the mission and keep on transmitting, or *move* to the sink and keep on transmitting, to facilitate delivery.

The nodes of the swarm need to know their current neighborhood, i.e. nodes within communication range, for selecting optimal relays for routing. As the topology continuously changes, the nodes periodically (every δ_{hello}) exchange hello packets $hello_{(i,*)}$ to communicate information that is relevant for routing, including node direction and speed. In the hello packet notation, $*$ symbol means that for node i, the destination of this hello packet was U, i.e., the packet was broadcast. When a node u receives a $hello_{(i,*)}$ from a node i, it stores the received information in its neighborhood map H_u. We denote with $H_u[i]$ the last hello packet received by u from node i. To select the next relay, the source node determines the presence of nodes among its neighbors, which are approaching the sink. The source node selects the relay nodes whose next destination and current speed are such that they can perform a two way communication of data packet and acknowledge messages, before the mutual distance of the two nodes exceeds their transmission range and the two nodes cannot communicate with each other any longer. If such a node exists, the source node transmits the packets to the selected relay node. If the transmission is successfully acknowledged (through an *acknowledgment packet*, *ack*), the relay node is responsible of delivering the packet to the sink, while the source node can continue its monitoring task trajectory. If instead a good relay is not available, the source node makes a new relay selection and transmission attempt after a time

interval δ_k, which is assumed to be uniform for all the nodes, and it is used to let the node distinguish whether a communication failed or not. We set δ_k larger than the communication round trip time between any two positions in the field of interest.

To avoid endless re-transmission attempts, and packet loss, MAD employs a reinforcement learning approach to decide whether the node should resort to a movement-based delivery, i.e. a physical movement in the direction of the sink to facilitate packet delivery. Note that the position of the sink, hereby denoted with σ, is assumed to be known by all the nodes at the mission start. The procedures for selecting a suitable relay and the transmission policy (i.e., *stay* or *move*) are described in Sections 4.1.2 and 4.1.2, respectively.

Optimal Relay Selection

To find the optimal relay a node must be able to estimate the time needed by a packet to reach the relay. Assuming loose synchronization among nodes, we estimate the time $\delta_{(u,i)}$ for node u to send a packet to node i as the time between the instant at which the $hello_{(i,*)}$ was generated on node i and the instant at which node u received it. Node u can then estimate the transmission error probability toward node i, denoted as $e_{(u,i)}(t) \in [0,1]$, by considering the percentage of packets sent to i that has not been acknowledged by time t. Set optimistically at the beginning of the mission to encourage exploration, this estimate is then updated over a sliding time window. Under the assumption that a suitable relay node must show a successful delivery probability of at least $\nu \in [0,1]$, a node u can compute the expected number of re-transmissions y, required to let node i receive the packet. Thus we can impose the probability of having at least one success out of y trials be greater than or equal to the hope ν, that is: $1 - e_{(u,i)}(t)^y \geq \nu$. From the above expression we can compute an upper-bound on the number of trials as: $y \leq \log(1-\nu) \setminus \log e_{(u,i)}(t)$. Considering the time δ_k between two consecutive transmission attempts, and given y trials, the expected waiting time before a successful delivery is given by $(y-1)\delta_k$. Notice that the value excludes the delay for the first transmission. The total expected time for a packet from u to be received at i, is defined as $\Delta_{(u,i)} \triangleq (y-1)\delta_k + \delta_{(u,i)}$, and it is used to estimate the time at which a relay node receives a packet, where, by definition, $\delta_k > 2\delta_{(u,i)} \; \forall \; u, i \in U$.

A node involved in the transmission of a packet selects its relay node on the basis of a *score based mechanism*. The score of a node i at time t considers the following information: position $x_i(t)$, speed $\dot{x}_i(t)$, transmission radius r_i, and next way-point $z_i(t)$. Every node is able to estimate the exact position of its neighbors exploiting the above features, extracted from the hello packets. We denote the time required for i to reach its next way-point $z_i(t)$ as $\delta_z^i(t) \triangleq \|z_i(t) - x_i(t)\| \,/\, \dot{x}_i(t)$, where $\|\cdot\|$ is the Euclidean norm. Upon transmission of a data packet $dpk_{(u,i)}$ at time t to a candidate relay i, node u estimates the current position of node i as $x_i(t)$. The estimate is accurate until node i does not reach its next target, advertised at time t_i in $hello_{(i,*)}$. Such an estimate is computed as follows:

$$x_i(t) = \begin{cases} x_i(t_i) + \dot{x}_i(t_i) \cdot (t - t_i) & \text{if } t - t_i \leq \delta_z^i(t_i) \\ z_i(t_i) & \text{otherwise} \end{cases} \tag{4.1}$$

Notice that, as we assume an online mission, the other nodes do not know the trajectory that node i will take after reaching its next target coordinates. Before assigning a score to every node, representing the quality of choosing it as a relay, a set of conditions should be met in order to avoid selecting nodes that will not ensure connectivity during the entire two-way communication, or that will take the packet farther from the sink. Node u checks on the following conditions for every node i for which it has received a hello packet $hello_{(i,*)}$ at any time $t_i \leq t$, the most recent one for every node.

Condition 4.1.1. *Node i is considered a relay if it will stay in the communication range of u until it would send $ack_{(i,u)}$ back to u:*

$$\begin{aligned} \|x_i(t + \Delta_{(u,i)} + \delta_{(i,u)}) - x_u(t + \Delta_{(u,i)} + \delta_{(i,u)})\| + \\ \dot{x}_i(t_i) \cdot ((t - t_i) - \delta_z^i(t_i)) \cdot \tau \leq \min(r_i, r_u) \end{aligned} \tag{4.2}$$

The inequality constraints the distance between the expected position of the source of the data packet u and the destination i, at the time of the ack packet reception (i.e., $t + \Delta_{(u,i)} + \delta_{(i,u)}$), plus a pessimistic estimate in case node i has passed its advertised target coordinates, to be greater than the minimum communication radius. In Equation 4.2, t is the time at which node u issues $dpk_{(u,i)}$, $\Delta_{(u,i)}$ is the expected time of reception at node i, $\delta_{(i,u)}$ is the time it takes for $ack_{(i,u)}$ to reach u. Notice that τ is 1 if $t - t_i \geq \delta_z^i(t_i)$ else 0, reflecting the fact that at time t node i has already reached its advertised way-point $z_i(t_i)$ or not, respectively. In the latter case, u makes a pessimistic estimate. An additive distance equal to the estimated distance traveled after having overcome z_i, is added to the distance of the two nodes. Notice that we consider only one attempt for i sending the ack for $dpk_{(u,i)}$, and for node u we do not add an additive distance in case it has overcome his $z_u(t)$.

Condition 4.1.2. *Node i will be closer than node u to the sink, at the moment of expected packet reception:*

$$\begin{aligned} \|x_i(t + \Delta_{(u,i)}) - \sigma\| + \dot{x}_i(t_i) \cdot ((t - t_i) - \delta_z^i(t_i)) \cdot \tau < \\ \|x_u(t + \Delta_{(u,i)}) - \sigma\| \end{aligned} \tag{4.3}$$

The inequality constraints the distance between the expected position of the destination of the data packet i and the source u, at the time of the data packet reception (i.e., $t + \Delta_{(u,i)}$), plus a pessimistic estimate in case node i has passed its advertised target coordinates, to be less than the distance between the expected position of the source node of the data packet and the sink. This condition guarantees greedy routing with preemptive nodes positioning estimate.

Figure 4.2 shows an example where a node u needs to choose a relay between nodes v and i. We consider two different scenarios. In the first scenario, we assume that a two way communication established by node u at time t, between both node v and i, it is expected to end at time $t' \triangleq t + \Delta_{(u,*)} + \delta_{(*,u)}$ where $*$ wildcards both v and i. Thus, Condition 4.1.1 is met for node v as it is still in the communication range $\|x_v(t') - x_u(t')\| \leq \min(r_i, r_u)$. Instead the position of node i at time t' may be out of the communication range of u, in the worst case, thus resulting in the violation of Condition 4.1.1. To define the second example scenario, using the

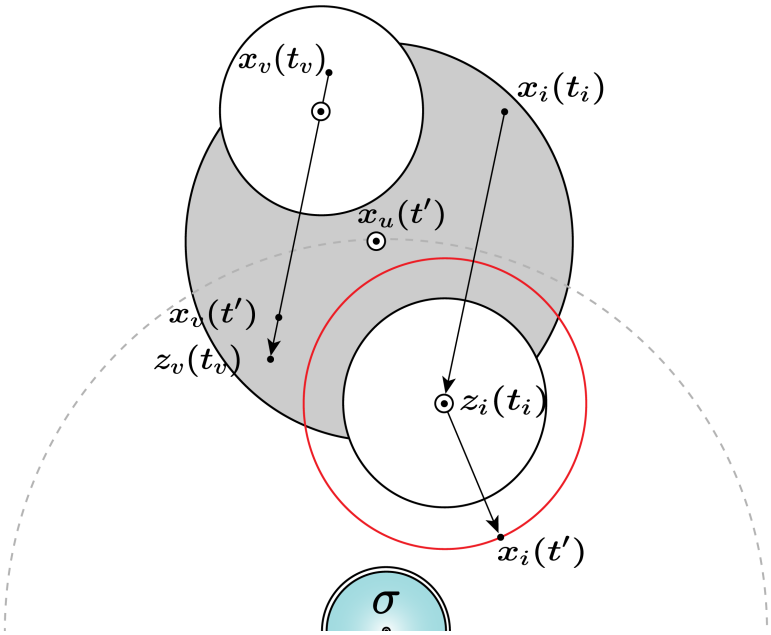


Figure 4.2. Optimal Relay Selection Conditions 4.1.1 and 4.1.2.

same Figure 4.2, we interpret the variable t' in a different manner, i.e., we consider $t' \triangleq t + \Delta_{(u,*)}$, to denote the moment of expected packet reception. At this time, both nodes v and i satisfy Condition 4.1.2, as their position at packet reception is closer to the sink than that of u. Nevertheless, node v will be chosen as it meets both the conditions.

We underline that Condition 4.1.2 ensures that a node u selects relay i only if i is at closer distance from the sink than u at the time of reception. However, once i receives the packet it can still move in a direction that increases the distance of the packet from the sink before its delivery. To prevent packet loss, we introduce a *fail safe mode* which ensures packet delivery, provided that the initial TTL of the packet is large enough to permit movement based delivery. The fail safe mode ensures that a node always opts for physical movement delivery whenever a packet residual lifetime is too close to the minimum time necessary to perform movement based delivery. We denote with $T_{res}(dpk, t)$, the residual lifetime of a packet at time t. If the packet dpk has been generated at time t_0, with time to live TTL, its residual lifetime is $T_{res}(dpk, t) = \text{TTL} - (t - t_0)$. Since node u is in position $x_u(t)$ at time t, it needs a time of at least $\delta_\sigma(u, x_u(t))$ to move and reach the sink, with $\delta_\sigma(u, x_u(t)) \triangleq \|x_u(t) - \sigma\|/\dot{x}_u(t)$.

Condition 4.1.3. *(Fail Safe Mode - Optional): A node u opts for physical delivery of dpk at time t whenever:*

$$0 < T_{res}(dpk, t) - \delta_\sigma(u, x_u(t)) - A < \delta_k, \tag{4.4}$$

where A is the time for the node actuators to modify the direction of the node to make it move towards the sink.

This condition is optional, and is meant to address critical applications. Theorem 4.1.1 characterizes the capability of MAD to guarantee packet delivery under fail safe mode.

Finally, the *score* of a node i is computed by u as follows:

$$\phi_{(u,i)}(t) \triangleq \|x_i(t + \Delta_{(u,i)}) - \sigma\| \tag{4.5}$$

The score is the distance of node i to the sink, at the time of the data packet reception. Now let $\Gamma_u^*(t)$ be the set of candidate relays of node u at time t i.e., the set of nodes that satisfy Conditions 4.1.1 and 4.1.2. The optimal relay of node u will be chosen as:

$$relay_u(t) \triangleq \underset{i \in \Gamma_u^*}{\operatorname{argmin}}\ \phi_{(u,i)}(t) \tag{4.6}$$

Node Transmission Policy Learning

MAD aims at meeting performance requirements given in terms of: *average packet delay* and *packet delivery ratio*. As shown in Figure 4.1, in case of neighbor device scarcity, or after a number of unsuccessful transmission attempts, MAD proactively moves disconnected nodes to seek for relays. MAD is used to let nodes learn an optimal policy that guides them through their mission so as to guarantee optimal performance requirements. The policy lets the nodes decide to either *stay* on the mission, in the hope to eventually meet a suitable relay (as defined in Section 4.1.2), or to *move* toward the sink. Purpose of the policy is to find a proper trade-off between application availability and delivery guarantee. Otherwise, (1) the node may end up moving too frequently towards the sink, even when the network deployment is quite dense, resulting in a very low packet delivery ratio, as most of the time the node will be far from its mission location; (2) the node may cause high average packet delay or may lose packets due to TTL expiration, resulting in a low packet delivery ratio.

In order to find this trade-off the nodes exploit a Reinforcement Learning (RL) approach. We will now give a brief introduction to the RL approach being used presenting the learning paradigm setup.

Reinforcement Learning Paradigm Setup

Through reinforcement learning, an agent gradually finds a decision policy on the basis of numerical reward signals it receives when performing allowed actions in the environment. Let $\mathcal{S}$ and $\mathcal{A}$ be the sets of states and actions, respectively. The goal of the agent is to learn a policy $\pi^*(\mathbf{s}) : \mathcal{S} \rightarrow \mathcal{A}$, i.e., a function mapping every state to the action that allows the agent to maximize the cumulative future discounted reward. Where the discount factor $\gamma \in [0, 1]$ weights future rewards. The policy is derived by means of a quality function $Q^*(\mathbf{s}) : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$, mapping the current state of the agent to quality values of the actions, according to the well known Bellman equation [31]. We use state of the art function approximators for estimating $Q^*(\mathbf{s})$, a neural network predicting action-values or quality values, we will thus refer to as Deep Q-Neural Network (DQN). We produce $Q(\mathbf{s}; \theta) \approx Q^*(\mathbf{s})$ with θ being the weights of the network. Given training instances: $\langle \mathbf{s}, \hat{\mathbf{s}}, a, r \rangle \in \mathcal{S}^2 \times \mathcal{A} \times \mathbb{R}$ the neural network is trained to minimize the loss computed as the squared error between the

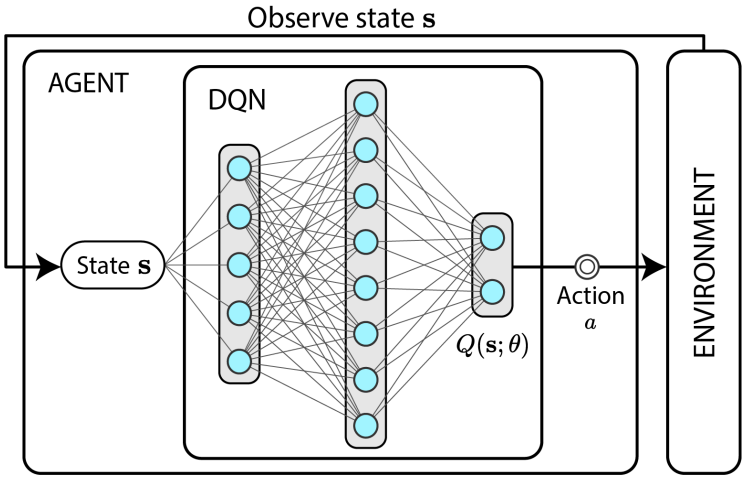


Figure 4.3. An RL Agent Querying a Pre-treined DQN.

target value of the action-value function and the *prediction* of the neural network as:

$$L(\theta) = \Big(\underbrace{r + \gamma \max_{a'} Q(\hat{\mathbf{s}}; \theta)[a']}_{\text{target}} - \underbrace{Q(\mathbf{s}; \theta)[a]}_{\text{prediction}} \Big)^2 \tag{4.7}$$

For MAD, the use of the DQN is twofold: (1) the DQN is trained to predict optimal action-values in a simulated environment, fed with data collected either beforehand or synthetically generated; (2) the pre-trained DQN is made available to all nodes in the network, that can query it to know the best action to take in the environment. Notice that, nodes could optionally use a pre-trained DQN to make some on-mission training through on field observations to possibly improve their performance. Figure 4.3 illustrates the second usage scenario mentioned above. At a new state observation, the agent queries the DQN to get an optimal action that it eventually executes in the environment.

Action Space The action space is $\mathcal{A} \triangleq \{move, stay\}$, where *move* forces the node to physically head towards the sink, thus suspending the monitoring mission, while the action *stay* keeps the node on the monitoring mission. As long as a node has packets to forward, every κ seconds it selects an action from the action space. Without loss of generality we assume that κ is a multiple of δ_k. This to ensure that the pursued action will be evaluated for a number of transmission attempts that will grant it to be evaluated fairly, even in case of failed transmission attempts. For the sake of brevity, throughout the section we adopt the notation $[K]$ to denote the set of integers $0, 1, \ldots, K$.

State Space The state features observed by a node are the following.

Sink Distance: It measures the distance of the node from the sink. It is denoted as $s^u_{\text{dist}}(t) : \mathbb{R} \to [M_d]$, where M_d is the linear distance between the sink and the farthest point from it, in the map.

Oldest Packet Age: It measures the age of the oldest packet in the buffer. It is denoted as $s^u_{\text{age}}(t) : \mathbb{R} \to [M_t]$, where M_t is the maximum TTL of the packets.

Local Density: It reflects the node density in the communication range of a node. Let $\Gamma_u(t)$ be the set of neighbors of u at time t, according the last received hello packets, we denote the local density as $s^u_{\text{dens}}(t) : \mathbb{R} \to [|U| - 1] \triangleq |\Gamma_u(t)|$.

Probability of Encounter: It measures the probability of meeting at least one node in the path towards the next target $z_u(t)$ of a node u. It is denoted as $s^u_{\text{meet}}(t) : \mathbb{R} \to [0, 1]$. The estimate is calculated by tessellating the area in a squared grid, whose side is set in proportion to the communication radius of the node. The frequency of observation of nodes in the tiles is used as an estimate of the probability of meeting at least one node in it, and is updated at every step. Let $\mathcal{C}$ be the set of all the tiles in the map, and $C \subseteq \mathcal{C}$ be the set of traversed tiles when heading from $x_i(t)$ to $z_i(t)$, the probability of meeting at least one node in the path is given by $1 - \prod_{c \in C}(1 - p_c)$, where p_c is the probability of meeting at least one node in the tile c. The estimate of the probability of meeting at least one node in a tile, is exchanged among the nodes and it is initialized with an optimistic value (high probability for all the tiles), to encourage exploration of the map in the initial steps of the simulation. For every tile, the nodes update their guess about the probability of meeting at least one node, when they fly over it, or when they receive more recent information from neighbor nodes. The update could be as simple as an average of the guesses weighted by the time elapsed between the node's actual guess and the neighbor's.

Mobility: It measures the variability of the nodes and consequently of the scenario. It is proportional to the rate of variation of the position of the neighbor nodes. It is denoted as $s^u_{\text{mob}}(t) : \mathbb{R} \to \mathbb{R}^+$. We will now give the analytical formulation for computing the mobility of a node. Let $v_{(u,i)}(t)$ be the function mapping time to the distance of node i to u:

$$v_{(u,i)}(t) \triangleq 1 - \frac{\|x_i(t) - x_u(t)\|}{r_u} \tag{4.8}$$

The variation of distance is computed as:

$$\dot{v}_{(u,i)}(t) = -\frac{1}{r_u}\left(\frac{x_i(t) - x_u(t)}{\|x_i(t) - x_u(t)\|} \cdot (\dot{x}_i(t) - \dot{x}_u(t))\right) \tag{4.9}$$

Thus we define the mobility measured at a node u as:

$$s^u_{\text{mob}}(t) \triangleq \sum_{i \in \Gamma_u(t)} |\dot{v}_{(u,i)}(t)| \tag{4.10}$$

where the derivative is flipped to the positive co-domain with the absolute value, since either raises (positive derivative) or drops (negative derivative) of distance denote a variation in the mobility of the node.

In order for the above cited features to be included in the state, they are evaluated instantaneously at the time they are queried, except for the mobility and local density, for which an average of the most recent time window is considered instead. We will generically refer to the state as the 5-tuple: $\mathbf{s} \triangleq \langle s_{\text{dist}}, s_{\text{age}}, s_{\text{dens}}, s_{\text{meet}}, s_{\text{mob}} \rangle \in \mathcal{S}$. All the co-domains of the features are normalized in the range [0,1] to make them have equal relevance for the learning process, thus the state space can be formally defined as: $\mathcal{S} \subseteq [0, 1]^5$.

Reward System When a node needs to transmit packets, it determines its state $\mathbf{s}$, and performs an action which results in a reward from the environment and a new landing state $\hat{\mathbf{s}}$. The reward function reflects what the agent should learn to do at best. The reward of action a (with $a = 0$ being the action *stay* and $a = 1$ the action *move*), while being in state $\mathbf{s}$ is the following:

$$r(\mathbf{s}, a) \triangleq \left(\frac{a\, s_{\text{dist}} + (1-a)\, s_{\text{meet}}}{s_{\text{age}}} \right)^{(1-2a)} \alpha^a \rho^{(1-a)} \tag{4.11}$$

which translates to: $\rho\, s_{\text{meet}}/s_{\text{age}}$ in case the node was staying on the mission, encouraging mission exploration in case of lower urgency of packet delivery and high probability of meeting at least a relay while heading to the target; $\alpha\, s_{\text{age}}/s_{\text{dist}}$ in case the node was moving, encouraging movement in case of high proximity to the sink and urgency of packet delivery. The parameters $\alpha, \rho \in [0, 1]$ reflect the importance of mission availability and delay respectively, and can be tuned according to the mission needs. For example, if delivering packets early is of uttermost urgency, regardless of mission unavailability, one can set α, ρ to 0 and 1, respectively, while intermediate values are used to obtain the most suitable trade-off between system responsiveness and mission availability.

Parameters Tuning

To automate parameter tuning, we introduce a Bayesian optimization framework. We recall that Bayesian optimization is a common technique used in Machine Learning to find the hyper-parameters $\mathbf{x}^*$ that maximize the unknown function value $f(\mathbf{x})$, which returns a noisy observation of the algorithm performance: $\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$ [149]. We consider α, ρ to obtain a function value that balances both delays and packet loss. We denote with P the set of all delivered packets, while $\widehat{P}$ is the set of lost packets, due to packet expiration or node unavailability. Therefore, the function value is $f(\mathbf{x}) = -\big(\sum_{pk \in P} \Delta_{pk} + |\widehat{P}| \cdot \omega \cdot M_t\big)$, where Δ_{pk} is the delivery time for packet pk. This function considers the cumulative delays of packets, where packets that were not received account for ω times the maximum delay M_t. Note that ω incorporates the roles of both parameters α and ρ, reducing the width of the hyper-parameters space and easing the algorithm setup according to the mission needs. The experiments proposed in Section 4.1.4 make use of the parameter values that maximize $f(\cdot)$, with $\omega = 1.5$.

MAD Algorithm

MAD is formally defined in Algorithm 7. The protocol is illustrated both in *training mode*, for off-line training of the DQN, and *querying mode* for querying a pre-trained DQN for on field usage of the protocol. Throughout the section we assume that node u is the node executing MAD at time t. Following an initialization phase, the algorithm goes through three parts we will now analyze in detail.

Part 1: Node identification and data reception (lines 3-11) Every node aims at being known by its neighbors and thus periodically, every δ_{hello} steps, it

Algorithm 7: MAD at node u

Initialize replay memory M and action-value function $Q : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ with random weights θ if *training mode* else if *testing mode* use input trained weights θ
for $t = 0, T$ **do**
  Make and broadcast packet $hello_{(u,*)}$ if $t \bmod \delta_{hello} = 0$
  Upon new packet reception pk
  if pk is hello packet $hello_{(i,*)}$ **then** $H_u[i] \leftarrow pk$
  else if *pk is data packet* $dpk_{(i,u)} : \langle id, \cdot \rangle$ **then**
    Enqueue data packet pk in B_u
    Make and unicast ack packet $ack_{(u,i)}$ for pk
  else if *pk is ack packet* $ack_{(i,u)} : \langle id, \cdot \rangle$ **then**
    Remove data packet of identifier id from B_u
    Set $z_u(t+1)$ to mission coordinates if B_u is empty
  if $t \, mod \, \kappa = 0$ *and* B_u *is not empty* **then**
    if *training mode* **then**
      Observe state $\hat{\mathbf{s}}$ and get reward $r(\mathbf{s}, a)$ as in Eq. 4.11
      Store $\langle \mathbf{s}, \hat{\mathbf{s}}, a, r \rangle$ in M and sample batch $m \in M$
      for $i : \langle \mathbf{s}, \hat{\mathbf{s}}, a, r \rangle \in m$ **do**

$$\mathbf{y}_i \leftarrow Q(\mathbf{s}; \theta)$$
$$\mathbf{y}_i[a] \leftarrow r + \gamma \max_{a' \in \mathcal{A}} Q(\hat{\mathbf{s}}; \theta)[a']$$

      Perform a gradient descent step on $(\mathbf{y}_i - Q(\mathbf{s}; \theta))^2$
      Set $\mathbf{s} \leftarrow \hat{\mathbf{s}}$ and action a:

$$a \leftarrow \begin{cases} \text{random from } \mathcal{A}, & \text{with probability } \varepsilon \\ \arg\max_{a' \in \mathcal{A}} Q(\mathbf{s}; \theta)[a'], & \text{otherwise} \end{cases}$$

    else if *querying mode* **then**
      Observe state $\mathbf{s}$ from the environment, set action a:

$$a \leftarrow \arg\max_{a' \in \mathcal{A}} Q(\mathbf{s}; \theta)[a']$$

  Set $z_u(t+1) \leftarrow \sigma$ if a is *move*, mission coord. if a is *stay*
  if $t \, mod \, \delta_k = 0$ *and* B_u *is not empty* **then**
    for $hello_{(i,*)} \in H_u$ **do**
      if *Condition 4.1.1 and 4.1.2 hold for node* i **then**
        $\Gamma_u^*(t) \leftarrow \Gamma_u^*(t) \cup \{i\}$
    if $\Gamma_u^*(t)$ *is not empty* **then**
      $m \leftarrow \arg\min_{i \in \Gamma_u^*(t)} \phi_u(i)$
      Make and unicast data packet $dpk_{(u,m)}$ for B_u

produces a hello packet and broadcasts it. Node u handles received packets according to their type. (1) A *Hello packet* is stored in the map H_u, mapping every node to the last hello message received.
(2) A *Data packet* $dpk_{(i,u)}$ is enqueued in B_u and a reception acknowledgment (referred to as ack packet) is prepared and unicast back to the originator of the data packet. Packets from the same sender will be acknowledged by a cumulative ack. (3) *Ack packets* are used to remove packets stored in B_u. Note that even the sink sends ack packets. If the queue happens to be empty, then the next mission coordinates $z_u(t+1)$ are updated, which forces the node to stay on the mission or to return to it in case it was performing a movement assisted delivery.

Part 2: Node DQN Training/Querying (lines 12-21) This part is executed at regular time intervals of length κ seconds, when the node has packets to send. It can be executed in two modalities, *training* or *querying*. (1) In the *training modality*, the node trains the weights θ of the action-value function $Q(\cdot\,;\theta)$, the DQN, using the reward signal received for the action pursued during the previous κ steps. In detail, the training is done over mini batches of training transitions $\langle \mathbf{s}, \hat{\mathbf{s}}, a, r \rangle$, sampled uniformly at random from the *replay memory* M. Such memory has a key role in *experience replay* [150], a technique utilized to help convergence when dealing with correlated data and non-stationary distributions. Going through all the transitions in the batch, the squared error in the prediction of quality values (i.e., the loss) is computed as in Equation 4.7 and the gradient of the loss is back-propagated in the DQN, exploiting the well known optimization algorithm known as Gradient descent. Following the training, the new action is chosen from the current state $\mathbf{s}$, either with probability ε from the set of possible actions or according to the action yielding the highest value. Notice that ε exponentially decays during the mission, significantly encouraging action exploration at the beginning of the training. (2) In the *querying modality*, a pre-trained DQN is taken as input, thus $Q(\cdot\,;\theta)$ outputs the optimal action-values and consequently the optimal action to execute from the currently observed state. The chosen action will determine if the agent will set its next target coordinates $z_u(t+1)$ to be the sink coordinates (a is *move*) or the next mission coordinates (a is *stay*).

Part 3: Node data send (lines 22-28) This part handles the optimal relay selection in case of non empty buffer, and is executed every δ_k steps. The node transmits all the packets in its buffer B_u. All the nearby nodes satisfying Condition 4.1.1 and 4.1.2 end up in the set $\Gamma_u^*(t)$. If such a set is non-empty, the node with the best score (Eq. 4.6) will be chosen as the best relay for data packet transmission. Notice that, the communication is executed also in the step immediately following the transition from empty B_u to non empty B_u (i.e., a new packet is generated or received), without waiting for δ_k steps for the first transmission.

4.1.3 Properties of MAD

In this section we formally analyze some of the properties of MAD including *termination*, and *time complexity*. To prove termination, we need to ensure that packets that are generated with a TTL that is sufficient to physically reach the sink onboard

a moving UAVs, are then delivered (with radio transmission or by physical device movements) without incurring unnecessary forwarding. Without such requirement, packets would either be delivered by means of radio transmission, or lost because of early expiration.

Theorem 4.1.1 (Termination)**.** *MAD guarantees packet delivery under the hypothesis that: (1) packets are generated with TTLs that are long enough to permit movement-based delivery from their location of generation and (2) the fail safe mode described in Condition 4.1.3 is enabled.*

Proof sketch. Let us consider a node u handling packet dpk whose residual lifetime is $T_{res}(dpk, t)$. We note that if $T_{res}(dpk, t) - \delta_\sigma(u, x_u(t)) - A < 0$, then MAD cannot guarantee mobile delivery. While hypothesis (1) guarantees that this never happens at the generation of a packet, we will show that this does not happen when the packet is sent to other nodes either, thanks to the fail safe mode of hypothesis (2). According to the fail safe mode condition, if $T_{res}(dpk, t) - \delta_\sigma(u, x_u(t)) - A < \delta_k$ then u takes the packet dpk to the sink with a physical movement. This of course ends up with a successful delivery provided that, by hypothesis, it is also $T_{res}(dpk, t) > \delta_\sigma(u, x_u(t)) + A$.

In contrast, radio transmissions can imply a loss of time during the delivery process. We want to exclude that radio transmissions causes the expiration of a packet, i.e. generate situations in which a packet is sent to a node i that has no sufficient time left to ensure physical delivery. We note that radio transmission is only enabled, under fail safe mode, when $T_{res}(dpk, t) - \delta_\sigma(u, x_u(t)) - A > \delta_k$. Let us consider the case of u transmitting dpk to node i. Notice that, by definition, δ_k is at least equal to the amount of time needed by u to determine whether the transmission to node i failed, in which case it has time to make a new evaluation or move towards the sink.

Then, we only need to show that if this transmission succeeds, the receiving node i has enough time to deliver the packet at least with a physical movement. Notice that i receives the packet at time $t + \delta_{(u,i)}$, with a residual lifetime of $T_{res}(dpk, t + \delta_{(u,i)})$. It then needs at least a time equal to $A + \delta_\sigma(i, x_i(t + \delta_{(u,i)}))$ to change direction and move towards the sink. Hence we must prove that $T_{res}(dpk, t + \delta_{(u,i)}) \geq A + \delta_\sigma(i, x_i(t + \delta_{(u,i)}))$. The above inequality can easily be derived by means of algebraic steps, given that (i) $T_{res}(dpk, t+\delta_{(u,i)}) = T_{res}(dpk, t) - \delta_{(u,i)}$ obtained considering the transmission time from u to i, (ii) that $T_{res}(dpk, t) > \delta_\sigma(u, x_u(t)) + A + \delta_k$ because of the fail safe mode, (iii) that $\delta_k > \delta_{(u,i)}$, and (iv) that $\delta_\sigma(i, x_i(t+\delta_{(u,i)})) < \delta_\sigma(u, x_u(t))$ because of Condition 4.1.2. □

Theorem 4.1.2 (Time complexity)**.** *The time complexity of MAD is $\mathcal{O}(N+|U|\cdot|\mathcal{C}|)$, where N is the maximum number of alive packets, $|U|$ is the number of nodes, and $|\mathcal{C}|$ is the number of tiles in the region of interest.*

Proof. In the following, we consider the complexity of MAD executed in querying mode (i.e., on field usage of the protocol), omitting the training process. For the nodes executing MAD, we assume that: broadcasting, unicasting a packet, choosing the target coordinates and querying the DQN, require constant time complexity.

The complexity of *part 1* is bounded by $\mathcal{O}(\log_2(N) + N)$. In this part, the most significant contribution to time complexity is given by the removal of the

packets from the buffer (***line 10***) upon reception of ack messages. The buffer is sorted by time of packet arrival, thus removing a packet from it would require a fast search method, such as binary search, from which the $\log_2$ term. In the worst case $\min(N, B)$ packets could be removed, where B is the maximum buffer size. The complexity of *part 2* is $\mathcal{O}(|U| \cdot |\mathcal{C}|)$. In fact, computing the state observation (***line 20***) for the features: *local density* and *mobility*, requires iterating over all the neighbors of a node, $|U|$ in the worst case. Concerning the *probability of encounter*, we note that in the worst case it should be updated for every tile and every neighbor, thus resulting in at most $|U| \cdot |\mathcal{C}|$ updates. The complexity of *part 3* is instead $\mathcal{O}(|U| + N)$. In this part, optimal relay selection requires evaluating the score of at most $|U|$ nodes in $\Gamma_u^*(t)$ (***line 27***). Sending the data packets requires $\min(N, B)$ steps. In conclusion, the time complexity of the whole algorithm is $\mathcal{O}(N + |U| \cdot |\mathcal{C}|)$. □

4.1.4 Performance Evaluation

In this section we study the performance of MAD by means of extensive simulations. In a first set of experiments we study the key features of MAD protocol, to motivate the benefits of trading off device availability with delivery guarantees. For this purpose, in Section 4.1.4, we evaluate MAD with respect to: (1) a baseline MAD variant, where device movements for physical packet delivery are disabled (i.e., $\pi(\mathbf{s}) = stay$, $\forall\, \mathbf{s} \in \mathcal{S}$), hereafter referred to as MAD^s; (2) a baseline variant that employs movement-based delivery only, while actively transmitting along the path towards the sink (i.e., $\pi(\mathbf{s}) = move$, $\forall\, \mathbf{s} \in \mathcal{S}$), referred to as MAD^m; (3) a delivery approach that employs movement-based delivery only, shortly denoted as Move. In Section 4.1.4, we compare MAD with state-of-art approaches, namely QMR [138] and DTN [139], specifically designed for FANETs.

Simulation Setup

We simulated a squared monitoring area of $2.25\, km^2$ with a data collection sink at a side, and we assumed that the nodes never cross the boundaries. The nodes follow a random way-point mobility model [12]. According to real field tests with commercial WiFi drones, and in agreement with recent literature works [102], we consider a node transmission range of 200m. To simulate the channel error we adopt the Free-Space Path Loss (FSPL) model [151]. Where not otherwise stated, we consider unlimited buffer size and energy availability for the nodes. In the following, we investigate the impact of movement-assisted delivery in terms of mission unavailability and flight time. We execute 50 runs, considering 3 hours long missions. The error bars in the plots represent one standard deviation from the mean value. In the simulations, we consider a critical scenario with urgent communications, which can only tolerate a bounded delay, reflected in the TTL value of the packets that, where not otherwise stated, is set to 300 seconds. We consider an event generation rate of about 6 events per minute, modeling random events occurring throughout the area of interest.

As for the learning process, we trained an offline model for a wide range of operation scenarios (e.g., increasing speed, or number of nodes). Notice that offline learning is often used whenever system responsiveness is particularly critical [152–154] so as to ensure high performance from the earliest mission instants, and to reduce

the computational overhead on the devices. The DQN neural architecture is made of 5 input neurons as the number of state features presented in Section 4.1.2, 8 hidden layer neurons, and 2 output neurons as the number of actions (i.e., *move* and *stay*). The discount factor γ is set to 0.85. Finally, we set a re-transmission waiting time $\delta_k = 1.5$sec, making decisions every $\kappa = 15$sec, with a required delivery probability $\nu = 0.95$ and a hello packet rate of 0.75 pkt/sec.

Metrics

Through the next sections we evaluate the routing protocol performance in terms of the following metrics. The *average packet delay* reflects the average time elapsed from the generation of a packet to its delivery to the sink. In case of duplicate packets, it considers only the earliest delivery time. It is computed only for the packets that are actually delivered, i.e. expired packets do not affect the value of this metric but are accounted in the *packet delivery ratio*, i.e., the ratio between the number of packets which have been successfully delivered to the sink and the total number of generated packets. The latter metric, reflects the capability of the routing algorithm to successfully deliver packets to the destination, before they expire. However, it does not account for missed opportunities to generate useful monitoring packets, due to node movements. For MAD, whenever a node decides to move for communication purposes, it abandons the desired application trajectory, and misses events. Similarly, for DTN, nodes acting as ferries do not actively contribute to the monitoring task as they would in case of balanced load. To factor the node unavailability due to controlled movements for transmitting data, we consider another metric, hereafter called *movement overhead*. This metric measures the ratio between the time spent performing movement-based delivery and the total mission duration. Notice that, this metric implicitly reflects the energy spent by the nodes to physically deliver the packets as well as their availability for executing the application mission. Finally, the *packet overhead* is the average amount of unnecessary transmissions generated by a data packet (i.e. packet duplicates). The above mentioned metrics will be evaluated in the next sections in terms of the following independent variables: *number of UAVs* deployed to monitor the field of interest, *UAV speed*, and *packet deadline* (i.e., the packets' TTL).

Performance improvement of mobility assisted delivery

We now show the results of a first set of experiments providing evidence of the benefits of the movement assisted delivery of MAD, with respect to MAD^m, MAD^s and Move.

We evaluate the four approaches under two experimental settings: (1) fixed speed of every node set to 8m/s and varying number of nodes in the network, ranging from 10 to 40; (2) fixed size fleet of 15 nodes at varying node speed ranging from 3 to 20m/s.

Fig. 4.4 and Fig. 4.5 show the average packet delay under varying number of nodes and speed, respectively. With Move such a metric does not vary significantly when the number of nodes increases, since it represents the average traveling time from any point of the map to the sink. The delay instead decreases with increasing

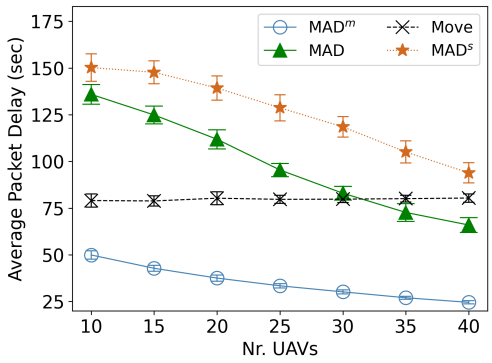


Figure 4.4. Average Packet Delay

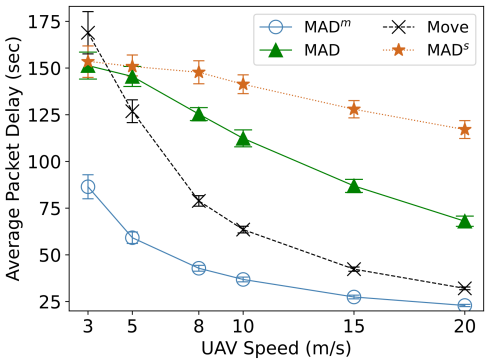


Figure 4.5. Average Packet Delay

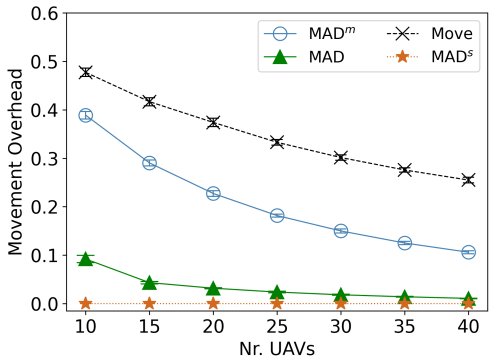


Figure 4.6. Movement Overhead

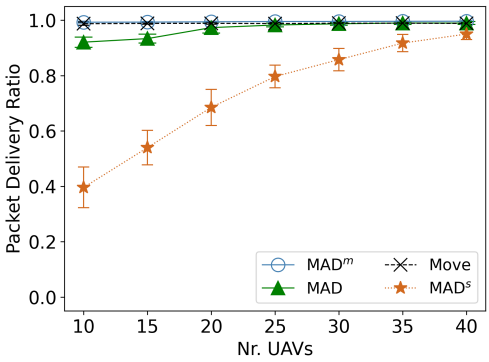


Figure 4.7. Packet Delivery Ratio

Scenario 1 : Baseline Comparison

speed as the average traveling time itself decreases. The average packet delay of MAD, MAD^s and MAD^m improves when the number of devices grows, as the higher device density enables more frequent multi-hop communications towards the sink, resulting in faster packet delivery. The same happens at increasing speed, as it takes less for the moving nodes with onboard packets to approach the sink. Clearly, MAD and MAD^s perform poorly when the number of devices is low (i.e., 10), as most of nodes are disconnected from the sink for a significant amount of time. However, the better performance of MAD^m and Move comes with a cost in terms of movement overhead. As shown in Fig. 4.6, MAD^m and Move spend respectively almost 40% and 50% of the observation time in movement for delivery. This result reflects their unavailability for the application mission, contrasting the apparently good performance in terms of average packet delay. With MAD^m and Move, the nodes do not execute their application mission, for a considerable percentage of the observation time. The overhead of MAD, instead, is negligible and improves significantly as the number of devices increases, showing the algorithm ability to successfully exploit the presence of nearby nodes, when available, to send data packets. In particular, MAD presents only 10% of movement overhead in the worst case (10 nodes), with a reduction of 75% with respect to MAD^m, while it achieves near zero overhead in the case of 40 nodes. Since MAD^s does not perform any movement-assisted delivery,

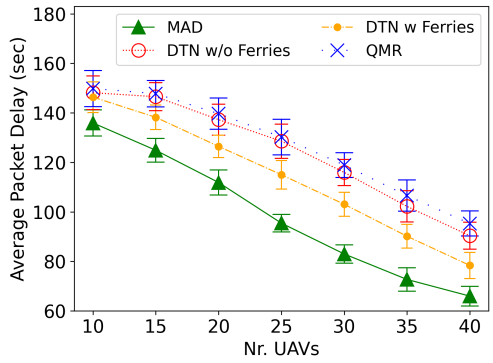


Figure 4.8. Average Packet Delay

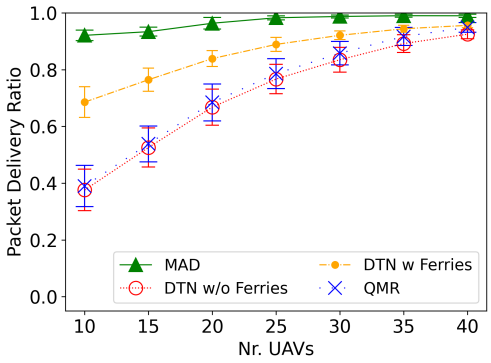


Figure 4.9. Packet Delivery Ratio

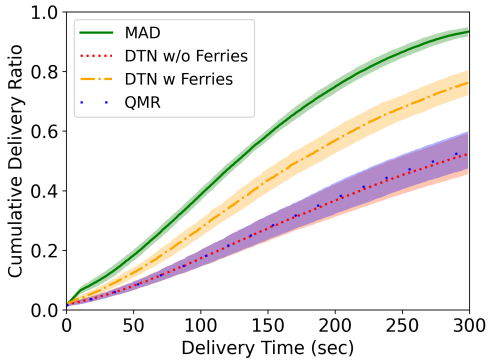


Figure 4.10. CDF of Delivery Time (8m/s)

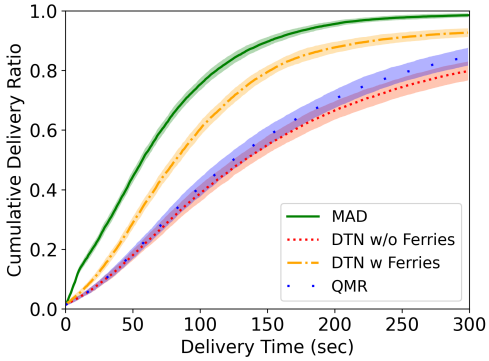


Figure 4.11. CDF of Delivery Time (20 m/s)

Scenario 2 : State-Of-Art Comparison

it shows no movement overhead, which as we have seen, comes at the cost of the highest delay among the compared algorithms.

Finally, Fig. 4.7 shows the packet delivery ratio by varying the number of nodes from 10 to 40. All movement-assisted delivery algorithms deliver nearly 100% of packets, independently of the number of nodes, while MAD^s delivers over 80% of packets only when the number of nodes is greater than 25 (enabling multi-hop communications towards the sink). We recall that this metric does not account for missed opportunities to generate useful monitoring packets, due to node movements. For example, with 10 nodes, MAD^m and Move participate in less than 60% of the mission, possibly losing 40% of the opportunities to generate, and therefore deliver, packets. MAD finds the best trade-off between delivering the most packets with the least possible delay and movement overhead.

These experiments confirm the validity of the hybrid approach which conjugates radio transmission with controlled movements for delivering packets to the sink, which is the key feature of MAD.

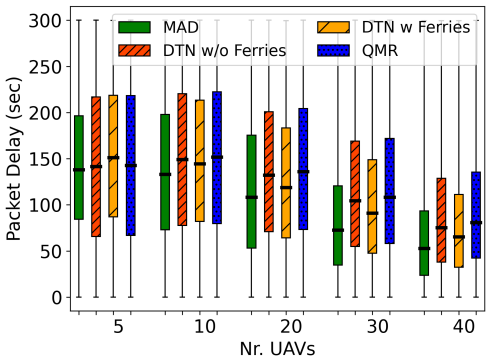


Figure 4.12. Packet Delay

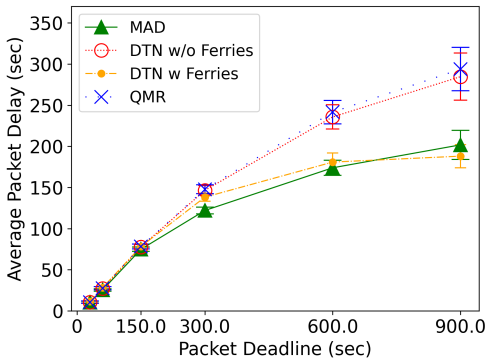


Figure 4.13. Average Packet Delay

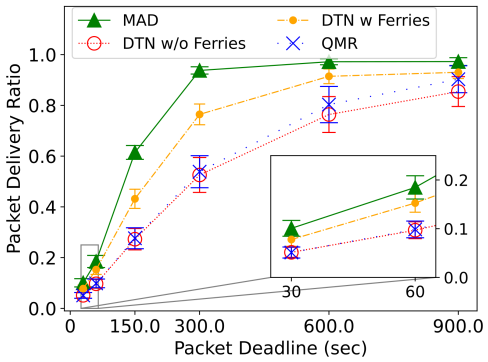


Figure 4.14. Packet Delivery Ratio

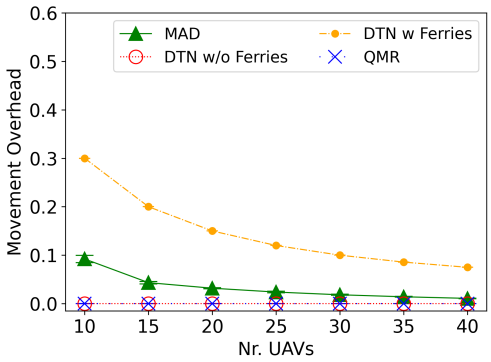


Figure 4.15. Movement Overhead

Scenario 2 : State-Of-Art Comparison

State-of-Art Comparison

We evaluate the performance of MAD against state-of-art approaches specifically designed for FANETs, namely QMR [138] and DTN [139]. In particular, while MAD can control the mobility of all the nodes of the fleet, DTN controls only a fixed number of dedicated ferry nodes, which move across the area to periodically carry packets to the sink, through dedicated routes. Therefore in our scenario we consider two variants of DTN: a) DTN-w/o-Ferries which does not have any dedicated ferry drones, and b) DTN-w-Ferries which uses 3 drones of the swarm as ferries to allow enough coverage of the area, as a higher number of ferries would result in poor resource usage and high protocol overhead. The trajectories of the ferries are the ones proposed in [139], with two sideward drones and one central drone. We first evaluate the protocols by setting the speed of every node to 8m/s and varying the number of network nodes, from 10 to 40.

Figure 4.8 shows that increasing the number of nodes allows for more chances of multi-hop delivery for all the algorithms, thus reducing the packet delay. However, MAD outperforms QMR and DTN-w/o-Ferries in almost all the considered range with a reduction of the average delivery time, which is higher than 25% when the number of devices is greater than 20. The similarity between DTN-w/o-Ferries and QMR is due to their geographical approach. In fact, DTN-w/o-Ferries in absence

of multi-hop paths towards the sink, selects the next relay based on a geographical approach like QMR.
MAD also overcomes DTN-w-Ferries of around 5% independently of the number of nodes. However we underline that this metric favors the benchmark algorithms because it only considers the delay of delivered packets. For this reason, we see that the performance gap between the three approaches is only moderate. We better analyze this aspect in the following figures.

Figure 4.9 shows how MAD outperforms all the protocols delivering almost 100% of the generated packets. When the number of devices is low (i.e., 10 drones) MAD doubles the delivery ratio of DTN-w/o-Ferries and QMR, while it surpasses DTN-w-Ferries of about 30%. When the network is more dense, MAD consistently shows the best performance, even if the delivery ratio of the other approaches increases, reducing the performance discrepancy (which they do at the expense of a corresponding increase in delivery time). The improvement of MAD over the other protocols is especially remarkable when the network density is scarce. This is the case where MAD benefits the most of movement based delivery.
Figure 4.10 and 4.11 show the cumulative distribution of delivery time, at two different speeds, with 15 nodes. The figure clearly shows that MAD delivers much more packets than any of the other algorithms. For instance, when the nodes fly at 8m/s, DTN-w-Ferries delivers 75% of the packets within their expiration time of 300 seconds. In contrast, MAD is able to deliver the same percentage within 200 seconds. DTN-w/o-Ferries and QMR only deliver around 50% of the packets within their expiration time of 300 seconds. Also, when the nodes fly at 20m/s, MAD outperforms all the approaches. While DTN-w-Ferries deliveries 90% of the packets within their expiration time of 300 seconds, MAD takes only half of the time for the same percentage.
To better analyze the delivery times of the packets, in Figure 4.12 we show a box-plot representing the quartiles of the packet delay distribution. The figure shows how MAD has the best performance almost always. With a low number of nodes, i.e. 10 nodes, MAD is slightly penalized in the first quartile (25% of packets) with respect to the other algorithms. However, as shown in Figure 4.9 it has the best packet delivery ratio, with possibly more packets in this first quartile, affecting this value. Increasing the number of nodes, the performance of MAD improves as well: it delivers most of the packets (i.e., 75%) within 125 and 90 seconds of their expiration time, for 30 and 40 nodes, respectively; while DTN-w-Ferries requires 150 and 110 seconds. Notice that, for all the algorithms, the minimum value is close to 0 seconds due to packets detected in proximity of the sink and delivered through a single hop communication; while the maximum value is 300 seconds representing packets delivered close to their expiration time. In Figure 4.13 and Figure 4.14 we evaluate the effect of varying the TTL (i.e., packet deadline) in a scenario with 15 nodes moving at 8m/s. This study also confirms the benefits of the movement assisted delivery performed by MAD. While MAD shows a comparable average packet delay with respect to DTN-w-Ferries, it consistently delivers more packets. In particular, even when the packet deadline is too tight (i.e., 30 and 60 seconds) MAD has the best performance. In terms of packet delivery ratio MAD is able to outperform DTN-w-Ferries of almost 50% when the TTL is 150 seconds.
Notice that in the lower range of the TTL values, all the algorithms show a very low

delivery ratio due to the scarcity of devices with respect to the size of the area. The performance of all the algorithms improves with increasing values. In fact, when the TTL is high, MAD and DTN-w-Ferries cope with the scarcity of devices by performing movement assisted delivery, while DTN-w/o-Ferries and QMR benefit from the additional time, which allows network topology changes and increases the chances of multi-hop delivery. In contrast, when the TTL is too low, MAD and DTN-w-Ferries do not have enough time to let the nodes physically carry the packets to their destination: a packet generated at farthest point from the sink may require up to 200 seconds to be physically delivered at 8m/s and the nodes are not dense enough to ensure radio delivery.
Figure 4.15 shows the Movement Overhead (% of time) spent by the drones to physically deliver the packets to the sink, by varying the number of devices. While MAD spends negligible time to physically delivery new packets, after a monitoring activity; DTN-w-Ferries dedicates a fixed amount of resources for mobility assisted delivery through ferries. In particular MAD has less than 10% of movement overhead with few drones (i.e., 10), outperforming DTN-w-Ferries of 300%, and it achieves near zero overhead with dense networks, as the communication capabilities reduce the need of physical delivery.
Is worth to notice that, this overhead reflects energy consumption of the drones for movement assisted delivery, as well as unavailability for the application mission (missed events). In fact, MAD nodes that actively move towards the sink temporally leave their mission tasks, while DTN nodes acting as ferries do not actively contribute to the monitoring task either resulting in unbalanced mission load (i.e., the drones actually used for monitoring may not be sufficient coverage of all the events).
At the expense of a small movement overhead, MAD outperforms all the previous solutions in all the considered metrics.

Finally, we discuss the packet overhead, for which we omit plots due to space limitations. Under a varying number of nodes, MAD outperforms all other approaches, in particular it produces a packet overhead that is 20% lower than the best of the other protocols (i.e., QMR). This trend confirms the benefits of the focused movement activity, and of the optimized relay selection of MAD, which are helpful in preventing the generation of unnecessary control messages, or of packet duplicates. Experiments conducted under varying node speed also confirm our conclusions, as MAD outperforms all the other algorithms. It is interesting to notice that when the speed increases, the packet overhead increases as well for all the algorithms, due to increased channel and transmission errors. Nevertheless, this increase is rather moderate for MAD, thanks to its optimal relay selection (see Section 4.1.2) which exploits a prediction of the future node positions.

We mention that we also evaluated MAD against two other algorithms designed for routing over MANETs, namely BATMAN [94] and GFG [140], which are considered promising for FANETs as well [7, 155]. These additional experiments were all favorable to MAD in all the performance metrics considered in this work. We do not include these results for space limitations.

4.1.5 Conclusions

In this work, we consider the problem of routing packets in a network of flying drones. We propose a routing algorithm, called MAD (*Mobility Assisted Delivery*), which exploits the device controllable mobility to facilitate network routing. MAD enables adaptive selection of the most suitable relay nodes and, based on local observations, resorts to a movement-based delivery (for instance, in case of neighbor device scarcity). We study the algorithm properties and, by means of extensive simulations, we show that MAD outperforms previous solutions in all the considered metrics, at the expense of a small loss in average device availability.

Chapter 5

Innovative Applications for Network of Drones

In the last chapters we mostly discussed the use of drones in safety-critical missions, however networks of drones are increasingly deployed also in other scenarios, including drone assisted cellular communication, airdropping of water and fertilizer, parcel delivery, agriculture-crop survey, or wildlife search [3, 5, 156–160],

In this Chapter we study three novel applications for UAVs networks, and we propose related solutions. In Section 5.1 we propose DANGER, a novel framework to build an emergency network of drones in case of disasters. Conversely from the existing work, DANGER can create a mesh network of drones which any WiFi smartphone can reach: it does not require a special application but the devices can easily access a simple web-application through a browser to chat with rescuers, with voice and videos.

In Section 5.2 we design DRUBER a distributed parcel delivery system aided by a blockchain framework. Druber proposes a fully distributed service based on a fleet of coordinated drones, belonging to multiple owners. To guarantee a trustable service Druber leverages blockchain features to develop and control the entire delivery chain. We show an impressive advantage of our platform regarding existing ground-based services in terms of service cost and parcel delivery time, at the expense of a negligible delay for the management of blockchain operations.

Finally, in Section 5.3 we discuss the adoption of aerial drones for Food Safety and Security. We show how a smartphone with a mobile Deep Learning application can detect diseases and provide sustainable food production in developing countries. We propose a MILP formulations and related algorithms to minimize and distribute smartphones to farmers, and cover all the region of interest. However, considering the limited amount of available smartphones in developing countries, we also discuss how drones can be used to cover farms more efficiently.

This chapter has been extracted from the works in [25, 26, 28, 29].

5.1 DANGER: a Drones Aided Network for Guiding Emergency and Rescue operations

Has now become more important than ever to guarantee an always present connectivity to users, especially in emergency scenarios. However, in case of a disaster, network infrastructures are often damaged, with consequent connectivity disruption, isolating users when are more in need for information and help. Drones may supply with a recovery network, thanks to their capabilities to provide network connectivity on the fly. However, users typically need special devices or applications to reach these networks, reducing their applicability and adoption.

To tackle this problem we present DANGER, a framework able to create a mesh network of drones, which can be reached by any WiFi smartphone. DANGER is highly flexible and does not require any special application: all connected devices can chat, with voice and videos, through a simple web-application. The DANGER network is completely distributed, can work even partitioned or in case of drones failures.

5.1.1 Motivation

One of the interesting technology advances in recent years is towards flying network devices. The vision is that Unmanned Aerial Vehicles (UAVs) or drones can be equipped with communication technologies that allow them to interact and provide network connectivity to unserved areas.

The benefits of a wireless network in the sky are clear. Such a network is far less expensive, far less disruptive and takes far less time to build than implementing an infrastructure-based network over areas where communications infrastructure currently does not exist or is disrupted. Above all, drones can reach harsh zones where land access is not possible, e.g., disaster areas [1]. Drones can provide communication capabilities to rescue teams and survivors, and provide the needed support during the overall post-disaster management.

While there has been substantial work on drones deployment [1, 161] practical issues of a real deployment have not been addressed. Most of the current proposals on emergency infrastructure-less networks have the goal of providing internet connection to all the nodes, i.e., drones and survivors' smartphones. To this end, they often require the user to install tailored mobile applications, which allow them to interact and participate in the mesh network [161] [162]. However, people typically do not install apps that may be useful in the event of a disaster. Hence, several users may remain isolated, unable to use the emergency network because they do not have the emergency app.

In this work we develop a framework that allows survivors to easily join and use the emergency network without any preinstalled app. We propose DANGER a infrastructure-less flying mesh network, which provides ubiquitous connection to all nodes, e.g., survivors' smartphones (see Figure 5.1). DANGER aims at providing interaction between rescuers and survivors (Fig 1.B), using a simple web application (Fig 1.C). The users and rescuers leverage the WiFi connection, provided by drones, to access a common distributed chat application. We believe that this approach can

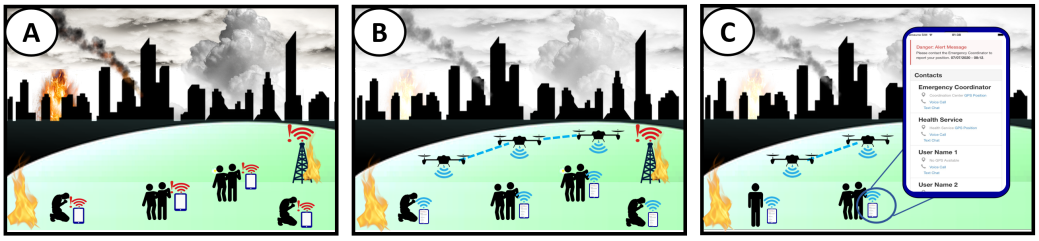


Figure 5.1. DANGER Architecture.

easily extend the applicability of emergency network to any scenario, without any special prefixed requirement, such as specific applications or procedures.

5.1.2 DANGER system design

DANGER brings network access to remote or isolated locations via a swarm of drones that allows users to chat, make voice and video calls through a drone emitted WiFi connection. Figure 5.2 shows the different drone's components.

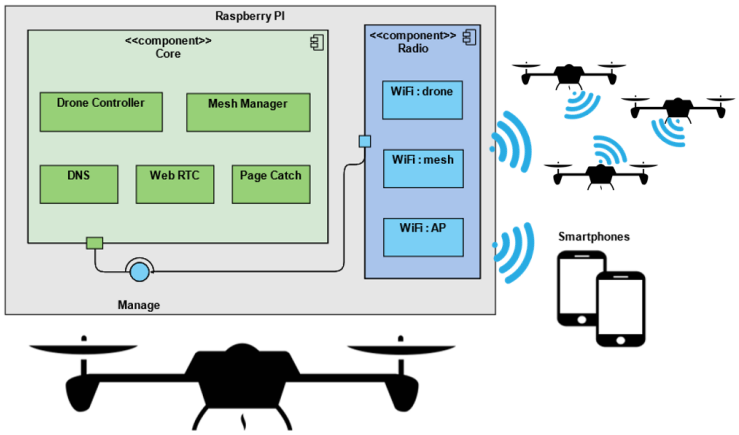


Figure 5.2. DANGER System.

Hardware Components. Each drone is equipped with a Raspberry Pi and multiple WiFi antennas. One antenna connects the Raspberry Pi to the drone, and allows to control the drone remotely or through a software intelligence which is executed on the Raspberry Pi. A second antenna enables drone mesh connection. Routing in the mesh implements the B.A.T.M.A.N. protocol [94], and allows drones to disconnect and reconnect, dynamically managing the network and assuring multi-hop connectivity. The third WiFi antenna creates an unprotected access point that is accessible by users mobiles. Notice that, if the drones have dedicated serial ports, the Raspberry Pi can directly use these ports without using a dedicated WiFi antenna. *Software Components.* The whole DANGER software is executed on board of the Raspberry Pi, mounted on drones. DANGER, as depicted in Figure 5.2, includes multiple components: i) users connectivity, which is composed by DNS, WebRTC and Page Catch; ii) Mesh Manager; iii) Drone Controller. Each drone exposes an open WiFi network called *DANGER*.

A survivor device can connect to DANGER network automatically, as it does not require any password. When WiFi is established, the DANGER framework creates a *catch-all* page at the address *10.0.0.1*, meaning that depending on the user device configuration it will automatically open that page or will redirect all the web requests to this page. This catch-all page, depicted in Figure 5.3, is managed through a DNS light server running on each Raspberry Pi. This page presents a broadcast section in the upper area of the screen, which shows a public chat where only the service manager can send information, and the list of users connected to DANGER. In case of *gateway drones*, meaning drones connected to external network or to the Internet, the Raspberry Pi may also have to run an instance of Web-RTC STUN and TURN servers. This would allow devices to contact other users connected to a different network (which have not joined DANGER). In this work however all devices can communicate only with other devices inside the same network, making STUN and TURN servers not required. The mesh manager is instead developed over B.A.T.M.A.N., and allows for multi-hop and fast drones disconnection and reconnection. The Drone Controller interacts with drones managing flight path. In this work we employ multiple Parrot AR Drone 2. Each one exposes a WiFi access point: connecting to it, an HTTP interface allows to control the drone. DANGER allows for both autonomous flight, for which we made a set of simulations on how to distribute the drones optimally on an area, and manual flight.

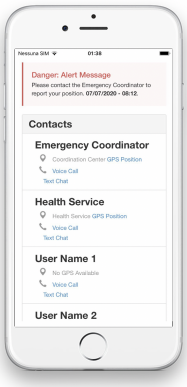


Figure 5.3. DANGER Web Interface.

5.1.3 Demonstration

In this work we showcase the use of DANGER with up to 4 flying drones and multiple smartphones. A real-field demo is performed in an open field of more than $130000m^2$, with multiple survivors equipped with a smartphone, which is disconnected from the cellular network. At the start of the demo the DANGER drones fly over the field. When a survivor is in the transmission range of a drone, her/his smartphone connects to the drone network. As soon as it gets connected, the smartphone presents the DANGER interface, with the possibility to contact the emergency coordinator and the other users. We showcase the communications through survivors and between a survivor and a rescuer.

5.2 DRUBER: A Trustable Decentralized Drone-based Delivery System

In the latest years the increasing number of online shoppers, 2 billion in 2019 [163], brought about enormous demand for fast and cheap systems to cope with the huge rate of parcel delivery requests. In dense urban areas, the delivery problem is exacerbated by traffic congestion, posing the challenge of maintaining a low environmental impact. Drones represent the fastest way to deliver parcels, thanks to their high mobility and automation. Several initiatives on the use of drones for CEP (Courier, Express and Parcel) delivery have been proposed by actors including Amazon, DHL, or UPS, just to name a few [164]. The effectiveness of drone delivery systems is assessed in [165], where more than 730 different use-cases in over 100 countries are presented. From the above discussion, it is clear that the use of drones is a cornerstone of the world of deliveries. Drones reduce the health risk for operators or postmen, as well as the tremendous environmental footprint produced by ground based deliveries [5]. In addition, drones can work automatically 24 hours a day, providing a faster service than the ground-based counterparts. Last but not least, law makers are gradually paying more attention to the growing industry of aerial drones, and are finally embracing the idea that drones are here to stay.

Recently, the interest in autonomous drone delivery received increasing attention because of the outbreak of COVID-19 emergency thanks to its potential to provide fast delivery without requiring any physical contact among human beings. Drones, as autonomous means of transportation, are effectively used in many critical scenarios to deliver parcels in emergency situations, such as earthquakes, pandemics, or floods [18,158,159]. Despite the wide interest on the use of drones for parcel delivery, the harsh reality is that only big companies can afford the cost for setting up and managing a widespread service infrastructure of drones and pick-up points.

To face this issue we envision the use of drones to enable a novel *fully decentralized* service for transportation of goods, based on a *distributed* cooperation model. The envisioned Druber service fleet is composed of drones that belong to private owners, i.e., people who earn money from providing the delivery service. Delivery of a parcel will be provided by multiple coordinated drones, with intermediate pit stops for battery replacement or drone-to-drone parcel handovers. Drones can literally straighten the delivery path of a parcel. Moreover, with Druber the delivery path will cover potentially unlimited areas, traversing entire urban and suburban regions, with no added delays but travel time.

The use of a *federated* approach eliminates the need for a single company investment and guarantees a quickly deployable, highly scalable, and inexpensive architecture, but introduces a problem of trust. Thus, the question we need to answer is whether users can trust private drone owners. Unlike centralized systems where registrations of devices, insertions of new requests, path planning, and parcel delivery can conceivably be handled by a trusted authority, we have no such guarantee in a federated and distributed environment. Thus, the challenge that we face is how to make Druber's operations trustable. Our key insight in this work is that Druber uses a simple but effective trick — all the operations should be public in order to have a clear feedback and guarantee a correct behavior from all the components.

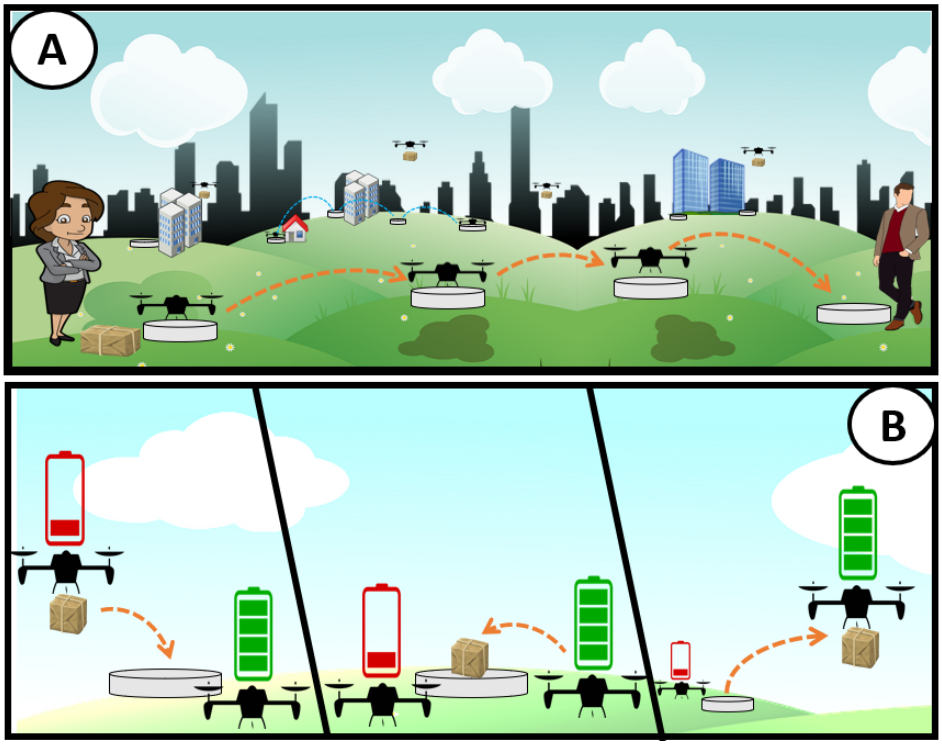


Figure 5.4. Druber Delivery System. The figure shows the Druber network in action, deployed along an urban area, and drones, which deliver parcels with handover at way-stations.

To this end, Druber leverages blockchain features to develop and control the entire delivery chain.
This approach allows for continuous validation of the drone work, making hard for drones to steal, disrupt deliveries, or obtain any sort of illicit profit from the provided service. We hereafter summarize the novelties of our work.

- First, we propose Druber, a federated decentralized system for parcel delivery in urban areas, composed of drones that belong to multiple private owners.
- Second, we address the trust issues, arisen from the fact that we deal with private drone-owners, who may behave in anomalous way or even maliciously. Our solution leverages blockchain features to develop and control the entire delivery chain, with moderate costs.
- Third, we evaluate the economic feasibility of the proposal, and evaluate quantitative and qualitative aspects.

5.2.1 Drones vs Truck delivery

The first question we ask is: are drones more efficient and convenient than ground vehicles for parcel delivery?

Time - Sending a package using drone shipping is clearly faster than using ground shipping. In case of traffic congestion, ground means may need a significant amount

of time to travel from the user pick-up point to the destination. Drones instead can reduce travel times thanks to the following aspects: i) they can be operational 24-hours a day (some drones can even fly in rainy or snowy conditions [166]); ii) they can follow the straight line path between departure and arrival location, without incurring any congestion; and iii) they can pick-up and deliver packets directly, without any temporary storage at warehouses.

Cost - Sending a package using drone shipping is also cheaper than using ground shipping. We compared the average cost of delivering a light parcel (<1Kg) with the Druber service and with ground transportation. For the computation of the average cost we considered an estimate of: 1) the cost of the drone, to be charged on the delivery price until break even; 2) the cost for the energy consumed by the drone; 3) the cost of periodic replacement of exhausted batteries; 4) the cost of the needed blockchain operations; and 5) the payout for the compensation of the drone owner. If the drone costs around 1500$, and the owner wants to repay it in one year, with an average flight distance of 10 km per day, then the fare for each kilometer has to include 0.4$ for the cost of the drone device. To estimate the cost for the energy consumed by the drone and periodic replacement of exhausted batteries, we refer to the work in [167] that quantifies this cost in 0.01$ per kilometer for a payload of 2Kg (the difference in delivering lighter payloads is negligible). To estimate the cost of blockchain operations, we consider the Ethereum Blockchain (see Section 5.2.3), taking into account the Ether value in April 2020 and the prices specified in the Ethereum Technical paper [168]. We developed an Ethereum smart contract which implements the Druber operations, and we estimated the cost of each hop. A delivery composed by 5 hops has a cost of about 0.25$, while a delivery of 20 hops costs about 1$. Finally, we consider the profit for the owner that we set to 0.1$. The average delivery cost for a parcel in New York, from Central Park to Wall Street (about 10km), is about 10$.

The cost of the same delivery service can be much higher if we consider ground based transportation. In New York, FedEx offers a service which guarantees delivery in 24h starting from 70$ [169].

5.2.2 System Overview

Druber is a novel service for transportation of goods, based on a distributed cooperation model. The envisioned Druber service fleet is composed of drones that belong to multiple private owners — people who earn money from providing the delivery service — and is managed by a decentralized system that oversees both customers and providers, as well as deliveries. The drones of the fleet execute their delivery tasks optimizing routes, service availability, delivery time, and sharing resources with each other. Private owners share the property of their drones or way-station, and make profits from their usage.
The system allows customers to request parcel deliveries, with virtually no distance limitations thanks to a parcel handover mechanism involving multiple cooperating drones. A parcel is picked up by a drone and delivered by multiple coordinated drones, with intermediate pit stops for battery replacement or for drone-to-drone parcel handover.

Use Case

Figure 5.4 shows the Druber network in action, deployed along an urban area, which is composed by: way-stations, for recharging and handover operations, and drones, flying above the city to deliver parcels. In particular, Figure 5.4.A shows a *mission*, i.e., a customer delivery request that is fulfilled by multiple drones, in a multi-hop manner. A drone picks up the parcel from the customer and flies until it reaches the next intermediate way-station, to handover the parcel and recharge its battery; finally, a drone delivers the parcel to the destination. Figure 5.4.B describes the handover mechanisms in details: first, a drone which is running out of energy lands at an intermediate way-station to release the parcel; then, a fully charged drone picks up the parcel and takes off, to continue the mission.

This framework extends the delivery paths to potentially unlimited areas, traversing entire urban and suburban regions, with no added delays but travel time.

System Components

The system is composed by both human and physical entities, namely *drone-owners*, *drones*, *way-stations*, and *end-users*.

Drone-Owners - Private people who own drones and way-stations and make them available to the delivery system, according to a federated approach.

Drones - We consider a set of private drones $\mathcal{U} = \{u_i\}_{i \in N}$, that pick-up, deliver and exchange parcels for the end-users. Each drone u_i is defined by a 6-tuple $\{p, r, a, b, oID, M\}$, where p is the current position, r is the drone range (the maximum distance that the drone can cover with a full charge), a is the drone current autonomy (the distance that the drone can cover with the current charge), b is the drone way-station, oID is the owner identification number, and M is the mission, if any, corresponding to a delivery request of an end-user.

Way-Stations - Physical points where drones can automatically perform recharging, off-loading and hand-over operations. In particular, they allow drones to land and take-off, cooperate by exchanging parcels, wait for new missions, or to recharge themselves upon need. Way-stations are possibly located on top of the city roofs, or in parking lots.

End-Users - The customers of the system, who request parcel deliveries.

System operations

We envision four main operations in Druber.

The first operation is the *registration of way-stations and drones*. When a drone-owner wants to join Druber she has to register her drone and way-station, communicating position and drone specs. Drone-owners can also withdraw a drone or a way-station. For each new registration/withdrawal, Druber updates the map of connected drones and way-stations in the urban area, useful to compute the set of delivery paths.

The second operation consists in the *insertion of a new Mission request*. When an end-user wants to deliver a parcel, she contacts Druber to create a new *Mission*

request M, which is defined by the tuple $< s, d, t, m >$, where s is the source way-station, d is the destination way-station, t is the delivery deadline (i.e. the maximum time within which the parcel has to be delivered); and m is the maximum amount of money that the customer accepts to spend for the delivery. Once the mission is created, Druber verifies if it is able to satisfy it under the cost constraint m, and in case it notifies the users and performs the delivery.

The third operation is *path planning*, which is responsible for finding a feasible and reliable path for a parcel, from the source to the destination. When the distance between the source and the destination is too long for a single drone, a multi-hop path is computed. To find a path we consider a distributed path planning protocol that incrementally builds the path (if possible) by collecting drones' *commitments* for the mission.

The fourth operation is the actual *parcel delivery* through multiple drones and way-stations (Figure 5.4). Druber guarantees the parcel delivery by handling possible failures and guaranteeing the correct drone behavior. In particular, the system must guarantee that all the drones of path P reached an agreement (commit) before the delivery starts. In case of failures or commitment withdrawal, the system must restart the mission by computing a new feasible path or a recovery plan. Finally, the drones receive feedback to penalize/reward their behaviors, i.e., a fee is given or charged to the drone-owners.

Performing these operations on a federated and fully decentralized framework as with Druber, poses considerable challenges. Our proposal leverages blockchain features to develop and control the entire delivery chain. In particular, we use the blockchain for delivery/chain management, and the Delegation-Bchain for path planning. In the following we explain how we leverage blockchain technology to make each of the above operations trustable.

5.2.3 Making Druber trustable

We now give a brief introduction to the blockchain technology and then explain how to exploit it to make Druber trustable.

Background on Blockchain Blockchain is a data structure in which each entry, called "block", is sequentially and cryptographically linked to the previous one, so that none of them can be changed. The blockchain is based on the *distributed ledger* logic, which allows to track transactions in a distributed way. So far, the two most adopted blockchains are Bitcoin and Ethereum. The last is a public blockchain which also allows to deploy applications called "Smart Contracts". Once written in the blockchain, such applications cannot be modified, and are executed by a virtual machine running on all the nodes in the peer to peer network. A deeper explanation of "Smart Contracts" is available in [168]. Every account in the Ethereum Blockchain owns an amount of Ether (ETH), i.e., coins that can be exchanged with Dollars, exactly like Euros or Yuan. An account can obtain Ether mining, i.e. doing work for the blockchain, or receiving Ether from other accounts, in exchange of services or real money. Each time that a user wants to write something on the blockchain, or wants to run some code already written in the blockchain, she has to pay a variable amount of Ether to the peers in the network behind the blockchain. As every content

written on the blockchain is saved on all ledgers, writing on the blockchain is slow and has a cost.

Our proposal We propose Druber-BC, a framework based on the Ethereum Blockchain that is composed by two main smart contracts:

- a Mission-Broker Smart Contract (MB-SC), which allows end-users to ask the way-stations for deliveries.
- a Validator Smart Contract (V-SC), which validates a mission path and handles payments.

We now go through each Druber operation (Sect. 5.2.3) and discuss how to make them trustable.

Registration of way-stations and drones

When a drone-owner wants to participate in the system by making available a way-station and/or a drone, she must register them to the framework. In particular, in order to register a new way-station, she can initiate a transaction with the MB-SC as shown in Figure 5.5. She must provide: i) the position (coordinates); ii) the IP address of the way-station; and iii) a list of the drones available at the way-station. Instead, in order to register a new drone, she must provide: i) the home way-station and ii) the drone details, i.e. the energy availability. The oID is automatically derived from the Blockchain user account.
To register a drone or a way-station, the system requires a caution deposit, i.e., a certain amount of Ether. This encourages genuine behaviors and penalizes bad ones. The deposit is used in case of a failure during a mission, which may cause a loss for the system or the end-user. As the delivery is performed in a multi-hop manner, if a drone withdraws from the mission, the MB-SC can use the deposit in order to rollback the mission and pay the other users involved. Therefore, each drone will be able to take part only to missions whose cost is lower than or equal to the caution deposit, to guarantee a cost-free rollback. The deposit is released when the drone-owner retires her way-stations and drones from the system.

Once the way-station has been deployed, it notifies its neighbour way-stations, to cooperate in the deliveries. By using the MB-SC, a way-station enables a cooperation with: 1) the way-stations that can be reached by its drones, i.e. their distance is less than the drones flight autonomy; 2) the way-stations whose drones are able to reach its home position. Therefore, the subscription is performed in double way, and each time a way-station receives a mission request, it forwards it to all the subscribed way-stations.
This operation is performed by sending a subscription request directly to the other way-stations' IP.

Insertion of a new delivery request

When an end-user wants to deliver a parcel with Druber, she has to create a new *mission*, using a dedicated application, and then she has to send it to the MB-SC.

A mission is characterized by:

- the source and destination coordinates;
- the maximum cost that the end-user agreed to spend;
- the current timestamp;
- the end-user blockchain address;
- the receiver blockchain address;
- the cryptographic signature of the first four fields, generated with the end-user private key;
- a pseudo-unique ID;

Notice that, the current timestamp along with the pseudo-unique ID allow to uniquely identify the mission request, while the end-user blockchain address identifies the end-user who requested it.
Once the end-user has defined the mission, she sends it to the MB-SC, along with the maximum amount of ETH she wants to pay for that delivery. The MB-SC verifies the mission and assigns the pseudo-unique mission ID. The new mission is saved inside the MB-SC Smart Contract, as shown in Figure 5.5. If the MB-SC accepts the mission, it broadcasts the assigned mission ID to all the way-stations through the emission of an event, which is a function of the Ethereum Blockchain and costs a fee that is negligible in comparison with the cost of a delivery (as estimated in Sec. 5.2.1). Although the cost for creating a new mission is low, it discourages denial of service attacks to Druber aimed at creating fake missions.

All the way-stations, even those that cannot participate in the mission, will then receive the mission details. However, they should immediately drop the mission, as their commitment would be rejected by the V-SC.

The end-user interacts with a dedicated application that queries the MB-SC for all the way-stations that own a drone in reach of the source, and sends the mission request to all of them. When way-stations receive the request they verify it with the MB-SC, and then they start the path planning operations.

Path planning: Delegation-BChain

We propose the so called *Delegation-Bchain*, to guarantee trust of path planning in a fast and cheap way. In fact, we note that a solution that writes in blockchain every possible hop of the path would be too expensive.

The simple idea is to offload several operations from the main blockchain, and sign the path only at the end of the planning. Specifically, in order to reduce costs, we propose the Delegation-Bchain, which allows to compute parcel paths from sources to destinations offchain, and then sign the path in the blockchain. A Delegation-Bchain is a chain of contents signed by blockchain accounts. The goal of a Delegation-Bchain is to reach an agreement on a mission request among a number of users involved, and sign the agreement on the blockchain.

A Delegation-Bchain is defined by a tuple (M, C, V). A mission request M represents a drone mission, and is defined by a set of requirements, namely delivery time, budget, and (source, destination) couple. The set $C = \{c_1, c_2, \ldots\}$ contains

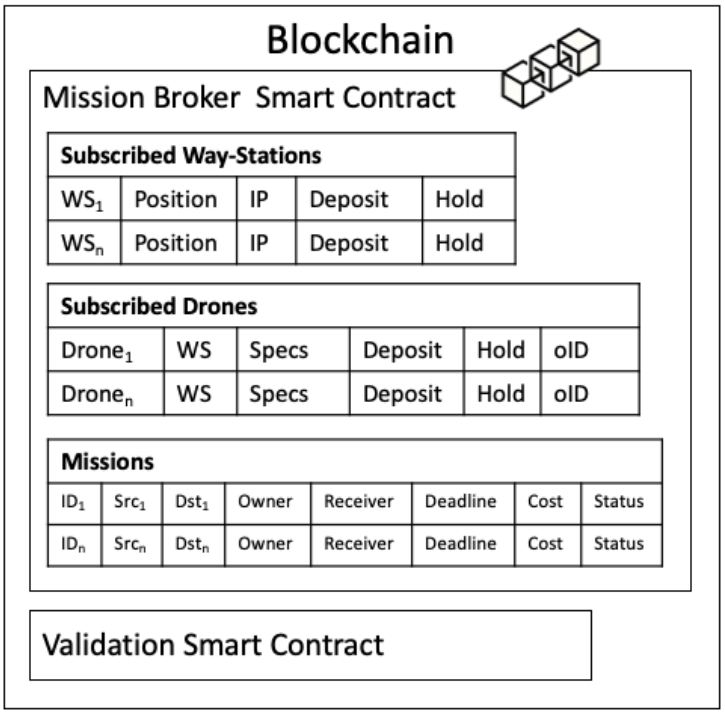


Figure 5.5. The Druber-BC entities.

the commitments of the drones who want to participate in the mission. While committing, the drone specifies the part of the mission it wants to perform, along with its constraints. In details, it provides i) the current way-station address; ii) the address of the next way-station; iii) the cost for the delivery; and iv) the current way-station signature of the Delegation-Bchain. The Validator V is a blockchain Smart Contract (V-SC) able to verify a Delegation-Bchain and to convert the commitments in escrows. With this conversion, each node who issued a commitment is definitely involved in the mission. For example, Alice wants to send a parcel to Bob and creates a new mission request (see Figure 5.6). Once way-station B_1 receives the new mission M from Alice, it creates a Delegation-Bchain with the mission M and its commitment (i.e., its promise to ship the parcel to a next way-station towards the destination). Commitments are signed with the private node key. Commitment signature represents the authorization of the drone to write its commitment on the blockchain. B_1 then forwards the Delegation-Bchain to nearby way-stations, in this case B_2 and B_3. The way-station B_2 does not have any drone available. However, it forwards the Delegation-Bchain to the near way-stations: they may have a drone that can go to B_2 to pick up the parcel. This is the case of B_4, which is also the destination way-station. Hence B_4 sends the Delegation-Bchain to the Validator, which verifies the feasibility of the commitments (i.e., whether all the signatures of the Delegation-Bchain are correct and the mission can be accomplished), and finally approves the Delegation-Bchain (i.e., it transforms all the accounts commitments in escrows). If a way-station sends a Delegation-Bchain whose commitments do not reach the destination or cost requirement is not met, the Validator will reject it. An

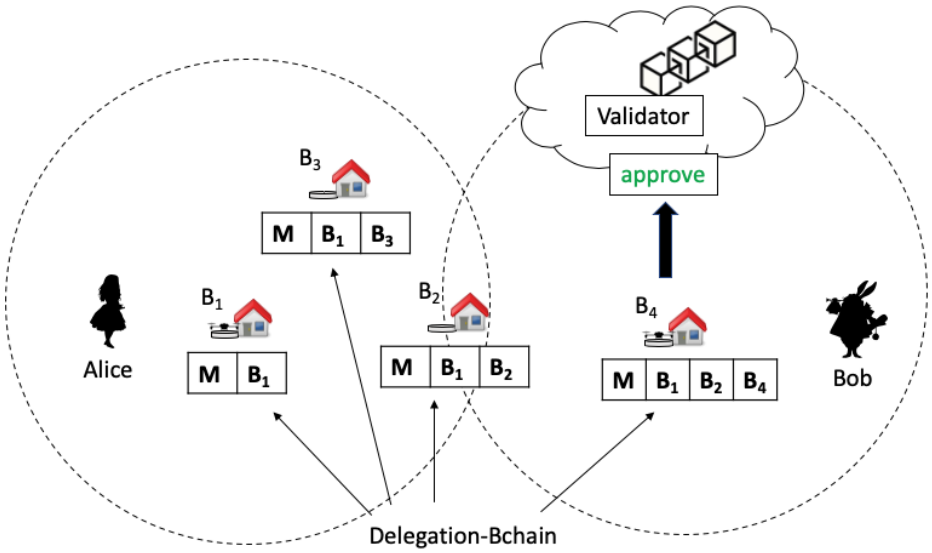


Figure 5.6. The Delegation-Bchain and the Validator. The Delegation-Bchain $(M, < B_1, B_3 >, V)$ is rejected, as B_3 cannot complete the delivery. The Delegation-Bchain $(M, < B_1, B_2, B_4 >, V)$ is accepted, as the drones of B_1 can take the parcel, perform an handover in B_2 with the B_4's drone, which can finally deliver the parcel.

intermediate way-station can also drop a Delegation-Bchain if the cost of the path built so far exceeds the mission budget.

The Validator is able to verify the contents of the Delegation-Bchain by checking that: i) all the commitments are correctly signed; and ii) the collected commitments are enough to satisfy the requirements of the mission request. If this verification succeeds the Validator performs all the operations written in the Delegation-Bchain, i.e., writes the sequence of commitments in the blockchain, and holds the deposits of involved drones and way-stations, so that they cannot accept multiple missions at the same time. Otherwise, as the Validator is a smart contract and its execution is atomic, if it fails in the middle of its execution (e.g. a drone is not available anymore), all the operations that it made before are rolled back. This may happen if a drone is already committed to another mission or if its owner has withdrawn it from Druber. In both cases the Validator can check that the deposit is absent or already on hold in the MB-SC. This approach is helpful in case of missions that require multiple tasks to be performed by different nodes, as commitments identify the hops in the path from source to destination. All the nodes interested in performing a part of the task required in the Delegation-Bchain can submit a "commitment" to the Delegation-Bchain. When one of the involved accounts realizes that the Delegation-Bchain contains enough commitments to complete the Request (e.g., a way-station has a drone that can complete the delivery), it can send the Delegation-Bchain to the Validator smart contract. In some cases, the mission may require every node interested in doing a task to "pay a commitment deposit". In this way if a node gives its commitment and withdraws it later, after the escrow, the Validator can hold the deposit. This deposit is used to refund all the other mission members that have been damaged by the retirement.

Parcel delivery

Once the path is successfully planned and the Delegation-Bchain is registered, the first drone on the path can carry the load and start the mission. At each handover the way-station that receives the parcel updates the parcel position in the MB-SC, and this information will be notified to the end-users until the delivery is completed. At the end of the mission the destination way-station updates the status on the MB-SC, and a fee is given or charged to the drone-owners. In fact, if a drone or a way-station aborts the mission, the MB-SC may hold their deposits in order to cover the cost of the mission restart or recovery. In general, for any problem during the parcel delivery, the current drone or a way-station, which is handling the parcel, will respond with their security deposit. Therefore the only way to gain money is to act correctly.

Advantages of using blockchain

The adoption of the blockchain as back-end brings the following significant advantages: i) as there is no central authority: the system can work without requiring any central entity and without a single stakeholder; ii) the system is able to offer guarantees between members, even if there is no trust between them; iii) the system is highly scalable and bottleneck free; iv) system operations are traceable and auditable; and v) the system is reliable because is fully developed on the blockchain, thus all the operations are tracked in an unmodifiable ledger.

5.2.4 Evaluation

Time Evaluation - We evaluate the time required for the creation of a valid Delegation-Bchain, which includes: i) the time required by Delegation-Bchain to travel from the source to the destination; ii) the time required by the blockchain to accept the Delegation-Bchain. In relation to i), each way-station requires no more than 1 second, meaning that for a delivery that involves 15 hops, the Delegation-Bchain takes 15 seconds to get to the destination. When the destination submits the Delegation-Bchain to the V-SC, if accepted, the system needs to wait for at least two blockchain's block closure before considering it as confirmed [168]. As for today, this time can be estimated in 30 seconds, a short time if we consider the advantages in terms of service cost and parcel delivery time.

Table 5.1. Qualitative Analysis

	Confidentiality	Integrity	Availability	Non-Repudiation
Druber	High	High	High	High
Standard Post	Medium	Low	Medium	Low
Dispatch rider	Medium	High	Low	Medium

Qualitative evaluation - We analyzed the Confidentiality, Integrity and Availability (CIA) features triad and Non-repudiation for Druber, standard post, and dispatch riders (see Table 5.1). Druber guarantees an unprecedented level of *confidentiality*, as any unfair entity would be discovered and excluded by the system. In terms of *integrity*, only a blockchain based system can guarantee that the system is

not altered by attacker or unfaithful employee. In terms of *availability*, Druber-BC runs on a distributed ledger, and in a complete peer-to-peer network. This makes it hard to interrupt the service, as there are no single points of failure. Moreover, in terms of *Non-repudiation*, once a mission is accepted by Druber the involved drones will work only and immediately for that order. Differently, both standard post or a dispatch rider may give different priority to orders.

5.2.5 Related Work

Drone based commercial deliveries are already a reality in some local implementations, such as the WakeMed's hospital where drones are used for ferrying medical samples [2], or Wing (a rib of Google's Alphabet) which recently launched the first residential drone-based delivery service to begin commercial operations in the United States [170]. However, all these initiatives provide unique drones' flights for a single delivery, where the same drone is in charge of carrying a single parcel from the source to the destination area.

On the security side, several works related to UAVs address the benefits of adopting blockchain, investigating new applications, from aerial traffic [171] to dam control [172], but with a different focus. Jensen et al. [173] propose a survey which explores how the blockchain can be applied to swarms of drones to provide defense against cyber attacks such as message spoofing and tampering. Aggarwal et al. [174] instead approach the Internet of Drones and propose a system, based on Ethereum, to guarantees anonymity, authentication, authorization and accountability in data dissemination. Mehta et al. [175] proposed a survey on the integration of UAV, blockchain and 5G, with the goal of guaranteeing several security aspects. They also propose a sample framework for drones based on blockchain for package delivery in the Industry 4.0 context.
However, this approach is not directly applicable to our scenario: performing the path planning operations in blockchain would introduce substantial overhead in terms of money and time, resulting inefficient. To the best of our knowledge Druber is the first attempt that offloads the most frequent operations to a Delegation-Bchain, leaving only the critical operations to blockchain.

5.2.6 Conclusions

In this work we propose Druber, a fully distributed parcel delivery service based on a federated fleet of coordinated drones. With Druber, the delivery of a parcel is performed by several drones, with intermediate pit stops for battery replacement or drone-to-drone parcel handovers. The use of a federated fleet guarantees a quickly deployable, highly scalable, and inexpensive architecture. However, the approach poses a number of issues to create a trustable service. In this direction, Druber leverages blockchain features to develop and control the entire delivery chain. Our evaluation shows the impressive advantage of our framework with respect to existing ground based services in terms of service cost and delivery time for light weight parcels.

5.3 Optimal deployment in crowd sensing for plant disease diagnosis in developing countries

The world population is estimated to grow and reach about 10 billions in 2050, significantly raising the demand for food [27]. Encouraging attention to the problem, the United Nations included the zero hunger goal [176] in their list of the 17 key challenges towards sustainable development. In most of the developing countries the poor availability of skilled personnel and supporting infrastructures make crop fields particularly vulnerable to the outbreak of plant diseases, due to spreading viruses and fungi, or to adverse environmental conditions, such as drought. In these countries new approaches are needed to mitigate the effects of climate changes and of spreading plant diseases, to produce sustainable food and eventually meet the UN zero hunger goal.

There are 550 million farms in the world and 83% are small, less than 2 hectares, family farms with little or no access to knowledge on increasing productivity. To face the lack of sufficient skilled personnel, government or agencies appoint expert people, with knowledge on plant diseases and treatments, called 'extension workers', to provide farmers with crop disease diagnosis. Unfortunately, extension workers are often not enough for the whole rural region. For example in our test-bed in Western Kenya, the numeric ratio between extension workers and farmers is about 1 : 5000 [177], while a much higher number, about 1 : 400, is needed to bring significant changes [178]. For several economical and political reasons it is difficult to change these numbers.

This brings about the need for smart technologies and tools that support the production of high-quality food. Agriculture is one of the fields in which the employment of IoT technology [179, 180], combined with crowd-sensing, may have a substantial impact. In this direction, the mobile application *PlantVillage Nuru* ("Nuru" means "light" in Swahili), provides an invaluable tool for early detection of plant diseases [181, 182]. Nuru analyzes plant images and uses a machine learning engine based on deep neural networks, which recognizes potential health issues. Nuru endowed smartphones can therefore be seen as key sensing components of a crowd-sensing framework where sensed data are collectively shared by the farmers.

However, most of the local farmers are too poor to afford a smartphone. These devices are typically provided by government agencies or by charity institutions, in a limited amount. A network of drones could easily cover much of the farms within few hours, requiring no human intervention with an expected lower cost from government agencies. However *PlantVillage Nuru* application is currently available only on smartphones, and the regions of interest have poor internet connectivity, resulting in a complex deployment. Therefore we consider only smartphones in our work, and we leave the use of drones as a future improvement (Section 5.3.6).

We propose a crowd-sensing framework, in which we hand out smartphones to some 'lead' farmers, who are in charge of providing the fundamental sensing activities of the framework. For the farmers, the opportunity to obtain a free smartphone constitutes an incentive [183, 184] to participate in the activities.

This framework is beneficial for two reasons. First, in the short term, the extensive use of the application provides farmers with the necessary information and

instructions to address upcoming issues in their plantations. This is of interest to individual farmers as well as to entire regions, where farmers individually addressing local plant issues, eventually stop the spread of harmful diseases which could compromise the crop yield and food production of the region. A broader perspective of crop status enables thus early discovery of diseases and prompt intervention for their treatment. Second, in a longer term, through the frequent use of the application, local farmers learn to recognize most plant diseases by themselves and treat them accordingly.

However, to ensure practical and widespread use of the proposed framework, we need to address several issues. A key problem is the minimization of the number of smartphones needed to ensure sufficient coverage of the farms of a region. Cost minimization, in fact, encourages investments from local governments and charities. Reduced deployment costs also enable the use of the proposed framework in wider areas, ensuring broader circulation of knowledge on plant diseases and on ways to increase productivity. Jointly with the described system sizing problem, we need to determine the most suitable farmers to receive the Nuru smartphone, and take the 'lead' role of their neighborhood. The lead farmer selection and related task assignment decisions must take account of farm locations and of the heterogeneous mobility patterns of people within the region.

To the best of our knowledge, ours is the first work to address this unique problem in the challenging scenario of developing countries, which requires ad-hoc problem formulations and specifically tailored solutions.

We propose two analytical models to minimize the number of required smartphones while providing sufficient geographical coverage, under the constraints posed by the environment, taking account of the limited controllability of the actions of participating farmers. Considering the high computational requirements of these models, we consider a hierarchical approach which splits the original problem into several smaller instances, when possible. Moreover, we contribute a greedy approach which trades off efficiency and scalability. We study the performance of the proposed solutions through extensive simulations. We compare our solutions with previously adopted approaches as well as with variants of state of the art solutions which consider similar problem formulations but in different contexts. Furthermore, we apply the proposed solutions in a real test-bed implementation in Western Kenya, in the Busia region, where crop loss due to untreated (or incorrectly treated) diseases was measured at a rate of about 40% (globally), with several zones with even $70 - 100\%$ of loss [185, 186].

We show how, thanks to the optimized deployment, the crowd-sensing framework ensures accuracy, responsiveness and cost, significantly outperforming current solutions where lead farmers are instead selected on the basis of people reputation or political motivations.

The major contributions of this work are the following:

- We introduce a crowd-sensing framework based on the use of mobile smartphones endowed with the application Nuru, to recognize plant health issues through image analysis;

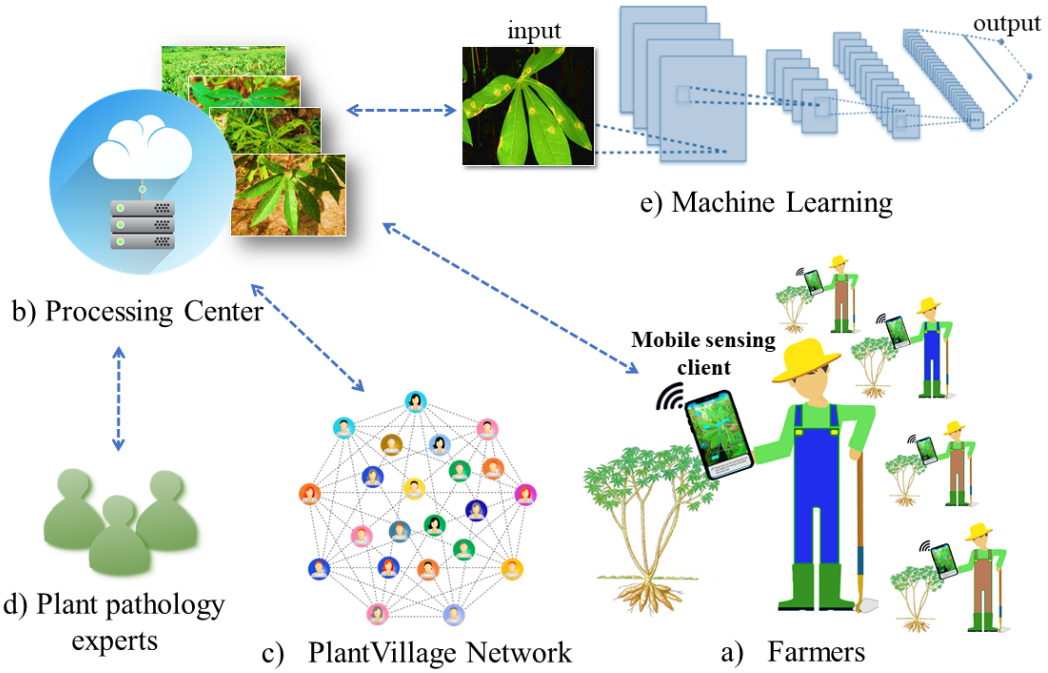


Figure 5.7. System Overview.

- We formulate the problem of joint device deployment and task assignment as an ILP. In addition, we propose a technique for a hierarchical reduction of the original problem instance into smaller ones;
- Considering the hardness of the deployment models, we propose a greedy approach to trade-off efficiency and scalability;
- We analytically study the proposed solutions, and evaluate their performance through simulations, against state of the art solutions to device deployment problems related to other application contexts;
- We show an implementation of the crowd-sensing framework in a real case test bed in Kenya, demonstrating practical applicability of the framework and its improvements with respect to currently adopted solutions.

5.3.1 System Overview

We propose the mobile crowd-sensing framework described in Fig 5.7. Through this framework, farmers actively use their smartphones to participate in the sensing operations and produce an updated view of the status of their crop field. The system is based on the components and agents described as follows. The farmers (a), endowed with Nuru capable smartphones, move along the crop fields located in the area of interest, performing the necessary sensing activity; the farmers use their devices to communicate with an online processing center (b) for information sharing, application updates, and to participate in social activities through the PlantVillage

social network (c). This social network allows farmers' communications and mutual feedbacks. Moreover, we observed that the farmers consider the PlantVillage social network as a way to gain social reputation and prestige, incentivizing their participation in the framework. The crowd-sensing framework also requires the participation of a crew of plant pathologists (d) for Q/A and labeling of disease images; the presence of these human experts is necessary for providing training sets and updating the machine learning system (e). Through the machine learning system the mobile application Nuru is able to successfully recognize plant diseases and suggests appropriate countermeasures.

The application Nuru has a twofold purpose. On the one hand, it identifies plants and related diseases providing diagnosis and suggesting countermeasures in the interest of individual farmers; on the other hand, it provides data for the system to reconstruct a global view of the fields within a region. Collected data are sent to the processing center as soon as connectivity is available. Upon reception of new data, the machine learning system is updated, as is the global view of the region crop fields. In this manner, the crowd-sensing framework provides health monitoring at a large scale, detecting the outbreak of new diseases and their spread throughout a region. The continuous update of the status of multiple crop fields in a region is essential to agencies and government entities to be able to design global campaigns to combat the spread of crop diseases at a region scale.

Figure 5.8. Farmer with phone examining cassava plant.

Nuru Smartphones

The proposed crowd-sensing framework utilizes basic smartphones as its key sensing devices. Notice that, while all smartphones are typically endowed with several types of sensors, being designed for low income countries, Nuru only requires a camera to

capture and process crop field images and monitor the crop health status. Figure 5.8 shows a farmer using the digital assistant to diagnose a plant of Cassava. The application allows farmers to access the four key functionalities of the framework shown in the app main menu in Figure 5.9. The farmers can: (1) participate into a farmer social network, in which they can *ask questions* to their peers; (2) chat with *human expert* plant pathologists; (3) access the plant database in the *knowledge library*; (4) and use the AI digital assistant for *immediate diagnosis*. The AI digital assistant is the main feature offered by the Nuru app. It is based on a Convolutional Neural Network (CNN) to detect and recognize possible diseases. In its current Tensorflow implementation [187], it is trained with a dataset of 2,400 cassava leaf photos, to successfully recognize 3 different diseases and 2 types of pest damage nutrition deficiency of cassava plants with about 70% accuracy for pronounced symptoms [188]. Further details on the AI implementation are available at [189, 190]. Figure 5.10 shows the diagnose mode of the application, in which a farmer is asked to point the phone camera at a cassava plant to identify possible diseases. Figure 5.11 shows the diagnosis results for an infected cassava plant, with a box around each leaf labeled with the predicted pathology.

To ensure successful image recognition in the face of possibly poor image quality, the model uses 8 seconds of video to have several redundant frames. In addition, the application provides farmers with a feedback on the quality of the captured images and may ask for additional shots if quality is insufficient. The effectiveness of the app was tested with low cost devices including the Samsung Galaxy S5 and the Tecno Camon CA6S [191, 192]. In our experimental setting, the combined approach of using redundant frames and repeated shots based on interactions with farmers, always produced images of good enough quality for the AI assistant to work.

As it was mainly built for developing countries, where smartphones are usually low cost and internet connection is not always available, the application works also when it is offline and image upload to the processing center can be postponed until internet connectivity becomes available.

For this reason, Nuru stores most of the data in the smartphone local storage, and compresses them to save space and reduce the future upload time.

Notifications and Feedback to farmers

The system uses the collected data to provide the farmers with information related to the status of their field as well as to region wide spreading diseases.
The outcome of the application analysis is used by local personnel and researchers to take appropriate countermeasures (e.g., to suggest specific medicines or pesticides) to heal diseases and boost crop productivity. The application also provides feedback charts, such as the one of Figure 5.12, representing the kilograms of dry mass per hectare during the last months. Furthermore, by evaluating the status of nearby farms the framework lets the lead farmers assess the necessities of other farmers and plan to provide assistance.

PlantVillage Nuru - Snapshots.

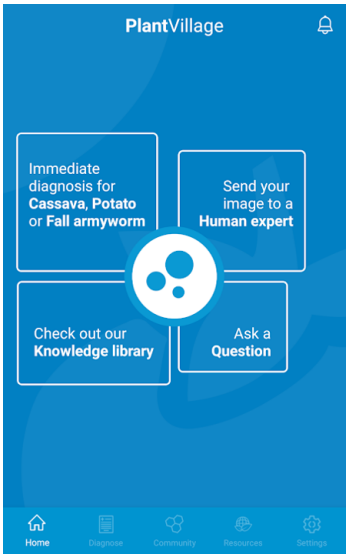


Figure 5.9. Main menu.

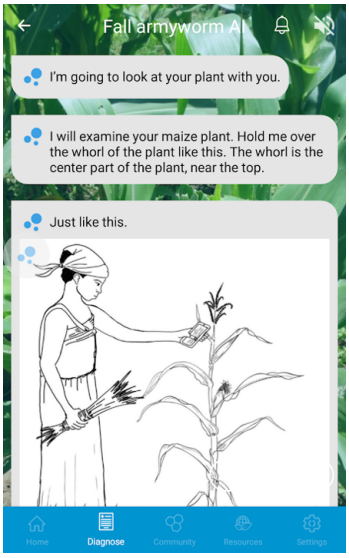


Figure 5.10. Diagnose mode.

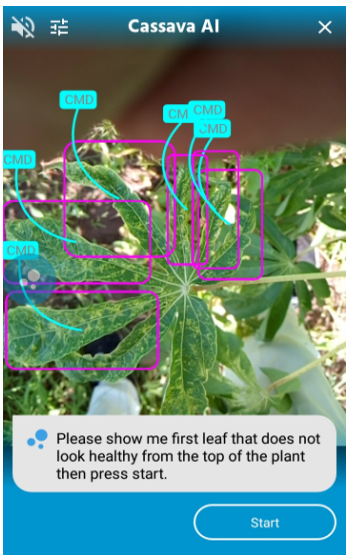


Figure 5.11. Diagnose result.

Incentives

We consider a participatory crowd-sensing framework where farmers use their smartphones to monitor their own crop field. In addition, lead farmers endowed with Nuru smartphones also travel around the neighbor farms to visit, help, and collect data related to a wider area. To encourage the user participation we considered several incentives.

Farmers participate enthusiastically in the crowd-sensing activities thanks to the following incentives:

- the possibility to diagnose diseases in their own plantations, thus preventing losses, providing healthy crops and better production;
- the free access to a crew of plant pathology experts and to the knowledge library;
- the possibility to help the neighbour farmers who do not own a Nuru smartphone;
- the access to the Nuru social network, in which farmers can interact, ask for help and share photos, which is gradually being seen as a way to gain social status, reputation and prestige among peers.

Moreover, the specific mechanism used to deliver smartphones to the farmer population, is also seen with great interest by the potential lead farmers. In fact, since the catchment area of the framework is mainly constituted by developing countries, smartphones are often given to farmers for free, or in exchange of their commitment

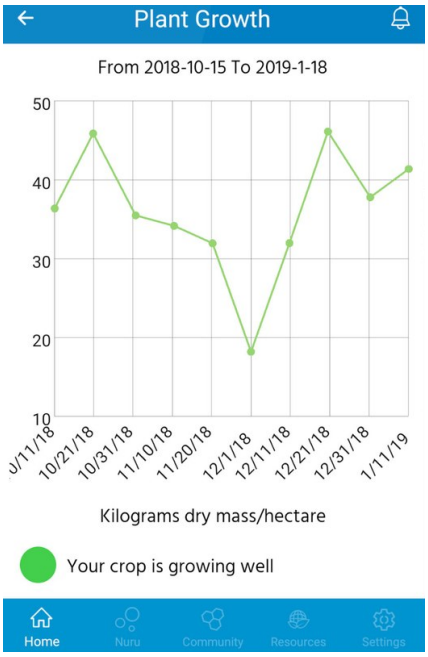


Figure 5.12. Framework feedback to farmers.

to participate in the framework activities. Therefore, another considerable incentive for farmers to participate in the crowd-sensing activities is the possibility to receive a free mobile device, with free internet connectivity.

Smartphone deployment

In developing countries farmers cannot usually afford the purchase of a smartphone and internet services. A small number of them receive these devices from their country governments with the intent of fostering digital education and consequently the use of digital services. Unfortunately, not all the farmers can be included in the country digitalization campaign. For example, for our test-bed in the Busia region, Kenya, the purchase of smartphones has been funded by the University of Penn State. Complimentary smartphones were handed out to 27 farmers in exchange of their participation in the test-bed activities.

To reduce the number of required smartphones, we used a Farmer-to-Farmer (F2F) collaboration model, similar to the one proposed in [193] where only some farmers (namely lead farmers) have access to the smartphones. These lead farmers help neighbor farmers by periodically visiting them, inspecting their crop fields, sensing and collecting data with their smartphones. The selection of lead farmers from the population plays a key role in the crowd-sensing framework. In fact, the adoption of an other than optimal deployment may result in zones with too many smartphones, i.e. where sensing is redundantly performed, while other areas, located too far away from their nearest lead-farmer home location, may be left uncovered.

We propose an optimized deployment of Nuru devices. Our approach produces a balanced distribution of Nuru smartphones over the region, reducing unnecessary redundancy. Moreover, it also assigns the minimum number of smartphones to

guarantee the required monitoring activity, resulting in a cost reduction to encourage governmental institutions to fund the system.

More specifically, we propose two analytical models, and a Greedy algorithm, to minimize the number of required smartphones while providing the necessary geographical coverage.

5.3.2 Smartphone Deployment and Monitoring Task Assignment

In table 5.2 we summarize the model notation. We recall that a lead farmer is a farmer endowed with the Nuru smartphone, who actively participates in the crowd-sensing activities, travelling around the region to visit neighbor farms, to provide auxiliary monitoring service to those farmers who do not have direct access to the framework, hereafter referred to as basic farmers. As Nuru smartphones are given to lead farmers, in the following we will interchangeably refer to the device deployment problem with the name of smartphone deployment or of lead farmer selection.

We propose an optimization model in which lead farmer selection and monitoring task assignment are addressed jointly. In fact, lead farmers must be selected so that they can provide monitoring service to a subset of farms close to their home locations, within distance and workload constraints.

We study two analytical models to guide decision making in terms of: minimum number of Nuru smartphones to deploy within the AoI, lead farmers selection, basic to lead farmer assignment, and lead farmer workload scheduling. These models enable a moderate cost deployment of the crowd-sensing framework while ensuring the required monitoring coverage and crop field inspection rate.

Notation	Description
AoI	Area of Interest
F_i	the i-th farm
$\mathcal{F}$	set of farms
k_i	required number of visits per farm i, per week
$\mathcal{L}$	potential lead farmers
ℓ_i	i-th potential lead farmer
M_i	maximum weekly workload of a lead farmer ℓ_i
w_{ij}	workload of ℓ_i to inspect F_j
c_{ij}	cost of assigning farm F_j to lead farmer ℓ_i
h_i	cost of selecting lead farmer ℓ_i
y_i	farmer deployment decision
x_{ij}	assignment decision of farm F_j to lead farmer ℓ_i
d_{ij}	distance between ℓ_i and farm F_j
v	average speed of a lead farmer
τ_j	inspection time requirement at farm $F_j \in \mathcal{F}$
b_i	maximum workload for a single inspection trip of ℓ_i

Table 5.2. Notation table

Let us consider an Area of Interest (AoI) including a number of farms. For simplicity we assume that the AoI is partitioned into a set $\mathcal{F}$ of disjoint farms F_i.

While the ultimate goal of the system is to ensure a yield increase of a given percentage (currently envisioned in terms of about 30%), we acknowledge that too many factors contribute to a complete mathematical formulation of this objective. Nevertheless we empirically recognized that a frequent enough activity of farm inspection, disease recognition and treatment, always translates into a rapid yield increase.

For this reason we require each farm F_i to be inspected with a guaranteed minimum frequency, hereby denoted with k_i, which typically depends on the scenario being considered. Notice that, while the observation period can be set to any time interval that is meaningful for the application, we consider the values of k_i as per week figures, $\forall F_i \in \mathcal{F}$. We discuss the tuning of this and other parameters in Sections 5.3.2 and 5.3.3.

We assume to have a set $\mathcal{L}$ of candidate lead farmers. Potentially every farmer can become a lead farmer if *selected*, endowed with a smartphone, and appropriately instructed. We assume that in order to turn a *candidate lead farmer* $\ell_i \in \mathcal{L}$ into an *actual lead farmer*, the system incurs a cost h_i (e.g. the smartphone cost or training cost). In theory, we may have as many lead farmers as basic ones, therefore we have $|\mathcal{L}| = |\mathcal{F}|$. Due to the objective of cost minimization, only the necessary lead farmers will be selected from the set $\mathcal{L}$.

The workload of a lead farmer is limited in time and space as follows. We consider a maximum lead farmer workload M_i, expressed as an upper bound on the frequency of visits of a lead farmer to neighbor farms, per week. We also consider an upper bound b_i to the maximum workload for a single inspection trip performed by farmer ℓ_i if selected as leader. Assuming also that, the visit of a farm j lasts about τ_j minutes, and that the lead farmer ℓ_i moves (walks) at speed v_i, the setting of b_i and M_i translates into an upper bound on the maximum distance that can be traversed by ℓ_i.

In details, consider the set $\mathcal{L} = \{\ell_1, \ldots, \ell_{|\mathcal{F}|}\}$ of potential lead farmers. We assume that lead farmers have a home position in the AoI, such that we can denote with d_{ij} the geographical distance between lead farmer $\ell_i \in \mathcal{L}$ and farm $F_j \in \mathcal{F}$. We also denote with τ_j the inspection time required for a visit to farm F_j. Therefore, a lead farmer ℓ_i can only visit the farms j for which $\frac{2 \cdot d_{ij}}{v_i} + \tau_j \leq b_i$, namely the farms that can be reached within a round trip time, including inspection time, lower than or equal to b_i (i.e., the upper-bound on the single trip workload).

We hereby formulate the two optimization models.

The first model, namely *Lead Farmer Selection and Trajectory Planning Problem* (LF-STPP), selects the lead farmers and decides the optimal trips to visit the nearby basic farmers. The second model, called *Lead Farmer Selection Problem* (LF-SP), is a simpler problem which only selects lead farmers and their associated basic farmers, but do not determine lead farmers' trips along the region as in the first model. In fact, LF-SP assumes a simplified route assignment according to which a lead farmer inspects the assigned farms one by one with a round trip back to her/his home location, in a star shaped route. This model does not enable any trip optimization but is not computationally intensive.

Figure 5.15 gives an example of the two models. The scenario in Figure 5.13 shows a solution to LF-STPP: the lead-farmer (0) visits the basic-farmers (1) and

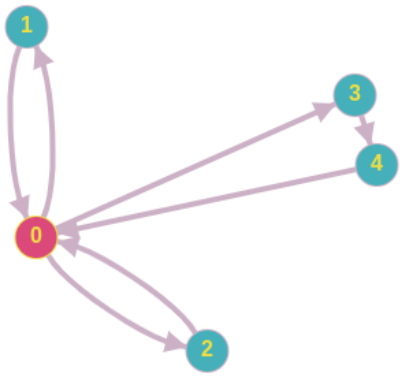


Figure 5.13. LF-STPP solution

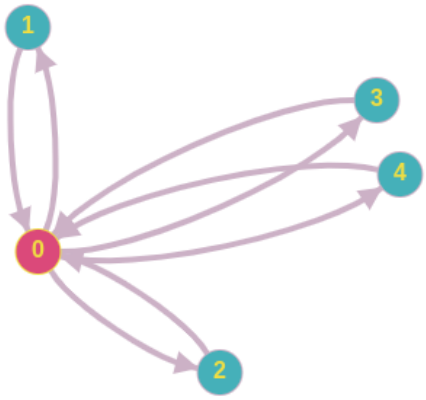


Figure 5.14. LF-SP solution

Figure 5.15. Lead farmer to neighbors path.

(2) in two consecutive trips; then it starts another trip to the basic-farmers (3) and (4), which are visited in the same optimized trip.

Figure 5.14, instead, shows the solution to LF-SP for the same instance. Differently from the previous solution, the lead-farmer (0) visits the basic-farmers (3), returns to her/his farm and then starts another trip to the basic-farmer (4). Even if LF-SP does not produce optimal routes (i.e., it wastes some of the time devoted by the lead farmer to the framework), its computational complexity is lower than the one of LF-STPP. Such a formulation also has several benefits and different fields of application, as discussed in 5.3.2.

Lead-Farmers Selection and Trajectory Planning

The first model, namely the *Lead Farmer Selection and Trajectory Planning Problem* (LF-STPP), selects the lead-farmers and assigns them a task scheduling in the form of optimized trips, potentially including several visits to some basic farmers within the same trip, when possible. It achieves the minimum cost of lead farmer selection, under the hard requirement that every farm F_j is inspected with guaranteed minimum frequency of k_j by a selected lead farmer. Such a minimum cost translates into a minimum number of selected lead farmers, if the individual cost of lead farmers is uniform, i.e., if $h_i = h, \forall \ell_i \in \mathcal{L}$.

We introduce the decision variables of the problem as follows. We denote with $y_i \in \{0,1\}$ the decision to select ($y_i = 1$) lead farmer ℓ_i or not ($y_i = 0$), for $i = 1, \ldots, |\mathcal{L}|$. Let $w \in [0, \mathcal{W}]$ be the index associated to the inspection paths of a lead-farmer, with W equal to the maximum allowed number of weekly trips W. We then define $x_{ij}^f(w) \in \{0,1\}$, with $i \neq j$ for all $\ell_f \in \mathcal{L}$, $F_i, F_j \in \mathcal{F}$ and $w \in \mathcal{W}$, the decision variable, to let the lead-farmer f move from farm i to farm j, exploring them in a sequence ($x_{ij}^f(w) = 1$) or not ($x_{ij}^f(w) = 0$) in the w-th trip.

Notice that, the correct value of $\mathcal{W}$ can be determined according to the values of the maximum weekly workload and inspection time of farms, or we can set $\mathcal{W} = |\mathcal{F}|$.

We recall that, given any two different farms $F_i, F_j \in \mathcal{F}$, d_{ij} is the distance that a lead-farmer needs to traverse to move from F_i to F_j, and τ_j is the necessary time to inspect the farm F_j. We model the time consumption of a lead-farmer ℓ_f along its path as the sum of the terms related to inspected farms and travelled distance. Therefore we model the time expenditure of ℓ_f to move from farm F_i to a different farm F_j, and visit farm F_j, as $\omega_{ij}^f \triangleq \frac{d_{ij}}{v_f} + \tau_j$ where v_f is the average speed of lead-farmer ℓ_f. We set $\omega_{if}^f \triangleq \frac{d_{ij}}{v_f}$ to exclude the lead-farmer's home farm from the cost of a trip. Notice that the appropriate setting of these weight parameters allows to model different lead-farmers' capabilities. We introduce c_{ij}^f as the cost of travelling from farm F_i to farm F_j for the lead farmer ℓ_f. This parameter may keep account of travel costs such as reward to the lead farmers or fuel cost.

The objective function of the model can be expressed as follows:

$$\min \sum_{\ell_f \in \mathcal{L}} h_f y_f + \sum_{i,j \in \mathcal{F}, \ell_f \in \mathcal{L}, w \in \mathcal{W}, i \neq j} c_{ij}^f \cdot x_{ij}^f(w), \tag{5.1}$$

which minimizes the cost of lead farmer selection and the lead-farmers' cost for the travels.

To guarantee that a basic farmer can be visited only by a lead-farmer, we introduce the following constraint:

$$\sum_{i,j \in \mathcal{F}, i \neq j, w \in |\mathcal{W}|} \frac{x_{ij}^f(w)}{|\mathcal{W}| \cdot |\mathcal{F}|^2} \leq y_f, \forall \ell_f \in \mathcal{L} \tag{5.2}$$

We also impose that each farm j be visited exactly k_j times by the same lead-farmer, unless she/he is a lead farmer:

$$\sum_{i \in \mathcal{F}, \ell_f \in \mathcal{L}, i \neq j, w \in |\mathcal{W}|} x_{ij}^f(w) = k_j \cdot (1 - y_j), \forall j \in \mathcal{F} \tag{5.3}$$

This constraint also prevents a same farm from being visited by multiple lead farmers.

We require that a lead farmer performs cyclic paths and prevent the formation of inefficient sub-cycles or unrealistic disconnected cycles in the solution by using the MZT technique [56] with auxiliary integer variables $z_i^f(w) \in \{1, \ldots, |\mathcal{F}|\}$. These variables are used to give an order to the all visited nodes, excluding the lead farmer home. To this purpose we introduce the constraints described in the following Equations 5.4 and 5.5:

$$\sum_{i \in \mathcal{F}, i \neq j} x_{ij}^f(w) = \sum_{i \in \mathcal{F}, i \neq j} x_{ji}^f(w), \forall j \in \mathcal{F}, \ell_f \in \mathcal{L},\ w \in |\mathcal{W}| \tag{5.4}$$

$$\begin{gathered} z_j^f(w) - z_i^f(w) \geq x_{ij}^f(w) + |\mathcal{F}| \cdot (x_{ij}^f(w) - 1) \\ \forall i, j \in \mathcal{F}, \ell_f \in \mathcal{L}, i \neq j,\ \forall w \in |\mathcal{W}| \end{gathered} \tag{5.5}$$

Then, we consider a capacity constraint to guarantee that a single trip does not exceed the upper bound on the maximum per trip cost b_f, for each lead farmer $\ell_f \in \mathcal{L}$ (this is meant to prevent too long uninterrupted routes in a single trip). This

constraint also ensures that a lead farmer is not associated with farms too far away from her/him, in terms of travel time:

$$\sum_{i,j\in\mathcal{F},i\neq j} \omega_{i,j}^{f} \cdot x_{ij}^{f}(w) \leq b_f, \forall \ell_f \in \mathcal{L},\ \forall w \in |\mathcal{W}| \tag{5.6}$$

Finally, we introduce the constraint of Equation (5.7) to guarantee that the total weekly workload of a lead farmer f does not exceed her/his availability M_f. Notice that M_f can be lower than $b_f \cdot |\mathcal{W}|$ (e.g., a farmer may have a daily/trip availability of 4-hours but may not want to work more than 12 hours in a week):

$$\sum_{i,j\in\mathcal{F},w\in\mathcal{W},i\neq j} \omega_{i,j}^{f} \cdot x_{ij}^{f}(w) \leq M_f \cdot y_j, \forall \ell_f \in \mathcal{L} \tag{5.7}$$

When employed, the constraint of Equation (5.7) along with Equation (5.3) makes Equation (5.2) unnecessary as (5.2) is always implied by them.

The overall model is defined in Problem 5.1. Notice that, the problem generalizes the (metric) uncapacitated facility location problem [194], which can be expressed by Equations 5.1, 5.2 and 5.3, and removing the other constraints. Such a problem is known to be NP-hard as it also generalizes the set cover problem[1].

$$\begin{aligned}
&\min \textstyle\sum_{\ell_f\in\mathcal{L}} h_f y_f + \sum_{i,j\in\mathcal{F},\ell_f\in\mathcal{L},w\in\mathcal{W},i\neq j} c_{ij}^{f} \cdot x_{ij}^{f}(w) && (a)\\
&s.t.\\
&\textstyle\sum_{i,j\in\mathcal{F},i\neq j,w\in|\mathcal{W}|} \frac{x_{ij}^{f}(w)}{|\mathcal{W}|\cdot|\mathcal{F}|^2} \leq y_f, \forall \ell_f \in \mathcal{L} && (b)\\
&\textstyle\sum_{i\in\mathcal{F},\ell_f\in\mathcal{L},i\neq j,w\in|\mathcal{W}|} x_{ij}^{f}(w) = k_j \cdot (1-y_j), \forall j \in \mathcal{F} && (c)\\
&\textstyle\sum_{i\in\mathcal{F},i\neq j} x_{ij}^{f}(w) = \sum_{i\in\mathcal{F},i\neq j} x_{ji}^{f}(w),\\
&\qquad \forall j \in \mathcal{F}, \ell_f \in \mathcal{L},\ w \in |\mathcal{W}| && (d)\\
&z_j^{f}(w) - z_i^{f}(w) \geq x_{ij}^{f}(w) + |\mathcal{F}| \cdot (x_{ij}^{f}(w) - 1)\\
&\qquad \forall i,j \in \mathcal{F}, \ell_f \in \mathcal{L}, i \neq j,\ \forall w \in |\mathcal{W}| && (e)\\
&\textstyle\sum_{i,j\in\mathcal{F},i\neq j} \omega_{i,j}^{f} \cdot x_{ij}^{f}(w) \leq b_f, \forall \ell_f \in \mathcal{L},\ \forall w \in |\mathcal{W}| && (f)\\
&\textstyle\sum_{i,j\in\mathcal{F},w\in\mathcal{W},i\neq j} \omega_{i,j}^{f} \cdot x_{ij}^{f}(w) \leq M_f \cdot y_j, \forall \ell_f \in \mathcal{L} && (g)\\
&y_f \in \{0,1\} && (h)\\
&x_{ij}^{f}(w) \in \{0,1\} && (i)\\
&z_i^{f}(w) \in \{1,\ldots,|\mathcal{F}|\} && (l)
\end{aligned}$$

Problem 5.1. LF-STPP - Lead Farmer Selection and Trajectory Planning problem

Lead farmer selection, a simplified problem

The second model, namely the *Lead Farmer Selection Problem* (LF-SP), selects a set of lead-farmers and assigns them a set of basic-farmers, without calculating paths, under the assumption that a lead farmer performs a unique visit to each basic farmer for each trip.

Notice that, this model is a special case of the previous Problem (5.1). When the lead farmers can visit only a basic farmer for each trip, the two models produce the same solution. This simplification may be justified, for example, when the lead farmers need to carry some load to basic farms, e.g. water, and must return to their

[1] A dual fitting algorithm is known to provide a 2-approximation to this formulation [194].

home location before making another visit. Furthermore, this simplified problem may be meaningful when a lead farmer trip limit b_f does not allow the consecutive visit of any two basic farmers in a single trip (e.g. $b_f < c_{fi} + \tau_i + c_{ij} + \tau_j + c_{jf}$, $\forall j, i, f \in \mathcal{F}$).

In these scenarios, the use of the second model, i.e. LF-SP, is preferable to the one of Problem 5.1, thanks to its simplicity and lower processing time requirements with respect to the generalized model.

While both problem formulations are NP-hard, we observe that LF-SP is still solvable in a reasonable processing time for medium instances of the problem. With about 100 farms, a Gurobi based solver [30] may find the optimal solution within few minutes. In contrast, due to the use of the MZT [56] constraints, the formulation of Problem 5.1, i.e. of LF-STPP, is particularly challenging in terms of processing time. For a limited scenario with only 60 farms, using a high performance computing architecture, namely a Lenovo X3550 M5, with 2 CPUs Intel(R) XEON(R) E5-2650 @ 2.20GHz with 16 cores each and 80 GB RAM [68], we experienced a computation time of dozens of hours.

The LF-SP problem can be formulated using the following decision variables: $y_i \in \{0, 1\}$ represents the decision to select ($y_i = 1$) lead farmer ℓ_i or not ($y_i = 0$), for $i = 1, \ldots, |\mathcal{L}|$; and $x_{ij} \in \{0, 1\}$, for all $\ell_i \in \mathcal{L}$ and $F_j \in \mathcal{F}$ is the variable that determines the assignment of farm F_j to the lead farmer ℓ_i ($x_{ij} = 1$) or not ($x_{ij} = 0$). We introduce c_{ij} as the cost of assigning farm F_j to lead farmer ℓ_i.

The objective function of the problem is expressed as follows:

$$\min \sum_{\ell_i \in \mathcal{L}} h_i \cdot y_i + \sum_{\ell_i \in \mathcal{L},\ F_j \in \mathcal{F}, i \neq j} c_{ij} \cdot x_{ij}. \tag{5.8}$$

which minimizes the deployment cost of lead farmers and their basic-farmers assignment cost.

The feasible set of solutions to the optimization of Equation (5.8) is determined by the following constraints. The first constraint imposes that each farm be assigned to one and only one lead farmer, if not to his-/herself in case of a lead farmer:

$$\sum_{\ell_i \in \mathcal{L}, i \neq j} x_{ij} = 1 - y_j, \forall F_j \in \mathcal{F}. \tag{5.9}$$

Second, an additional constraint requires that farms be only assigned to selected (actually deployed) lead farmers:

$$\sum_{j \in \mathcal{F}, j \neq i} \frac{x_{ij}}{|\mathcal{F}|} \leq y_i, \forall \ell_i \in \mathcal{L}. \tag{5.10}$$

This constraint ensures that if lead farmer ℓ_i is not selected, it is not assigned to farm F_j for inspection. The constraint is valid for any farm $\ell_i \in \mathcal{L}$.

Notice that this problem is also NP-hard as the equations (5.8), (5.9), and (5.10) alone, give the formulation of the (metric) uncapacitated facility location problem [194], which is known to be NP-hard.

We now consider the capacity constraint, by expressing the maximum allowed weekly workload of a lead farmer in terms of time spent for inspections and travels to farm. We recall that the maximum weekly load of each farmer ℓ_i is M_i, the

maximum trip load is b_i, and that we denote with τ_j the visit requirement of farm F_j. The capacity constraints take account of both the round trip time, from the lead farmer location to a farm and back, which, assuming a constant speed v, can roughly be assumed proportional to $(2d_{ij}/v)$ and the inspection time τ_j.

Therefore, we denote with w_{ij} the workload of a single visit of ℓ_i to farm F_j: $w_{ij} \triangleq 2d_{ij}/v_i + \tau_j$. The lead farmer weekly capacity constraint can be modelled as follows:

$$\sum_{F_j \in \mathcal{F}, j \neq i} w_{ij} \cdot k_j \cdot x_{ij} \leq M_i \cdot y_i, \ \forall \ell_i \in \mathcal{L}. \tag{5.11}$$

As for the previous Problem 5.1, the value k_j is the number of required visits for a week for the farm j. In this equation it is used as a multiplier for the cost of a single lead-farmer visit and, thus, it is taken in account into the lead-farmer weekly workload.

Notice that the constraint of Equation (5.11) along with the Equation (5.9) makes Equation (5.10) unnecessary as Equation (5.10) is always implied by them.

Finally, we keep account of the maximum trip time constraint b_i, by allowing each farm to be assigned to only lead farmers with an inspection cost lower than b_i. Such a requirement implies:

$$w_{ij} \cdot x_{ij} \leq b_i \cdot y_i, \ \forall \ell_i \in \mathcal{L}, \ \forall j \in \mathcal{F}, j \neq i. \tag{5.12}$$

The optimization problem is then a *Bounded Distance, Capacitated Facility Location Problem*, formally defined in Problem 5.2.

$$\begin{array}{ll}
\min \sum_{\ell_i \in \mathcal{L}} h_i \cdot y_i + \sum_{\ell_i \in \mathcal{L}, \ F_j \in \mathcal{F}, i \neq j} c_{ij} \cdot x_{ij} & (a) \\
s.t. & \\
\sum_{\ell_i \in \mathcal{L}, i \neq j} x_{ij} = 1 - y_j, \forall F_j \in \mathcal{F} & (b) \\
\sum_{j \in \mathcal{F}, j \neq i} x_{ij} / |\mathcal{F}| \leq y_i, \forall \ell_i \in \mathcal{L} & (c) \\
\sum_{F_j \in \mathcal{F}, j \neq i} w_{ij} \cdot k_j \cdot x_{ij} \leq M_i \cdot y_i, \ \forall \ell_i \in \mathcal{L} & (d) \\
w_{ij} \cdot x_{ij} \leq b_i \cdot y_i, \ \forall \ell_i \in \mathcal{L}, \ \forall j \in \mathcal{F}, j \neq i & (e) \\
x_{ij}, y_i \in \{0, 1\}, \ \forall \ell_i \in \mathcal{L}, F_j \in \mathcal{F} & (f)
\end{array}$$

Problem 5.2. LF-SP - Lead Farmer Selection Problem.

Model tuning

The proposed deployment models require several parameters (as shown in table 5.2) to be tuned before we can use them to configure the crowd-sensing framework. For example, assuming to work with weekly workloads, we need to know the position of farmer F_i, the required number of visit k_i, the maximum weekly workload M_i for a lead farmer, the mobility of farmers, expressed in terms of average speed v, and the inspection time requirement at each farm τ_j.

While the farmers' positions must be collected by human personnel, or using geographical data and images, the other parameters can be estimated using previous data (e.g. from similar scenarios) or by submitting forms to the farmers before deploying the system.

In the section 5.3.3 we discuss and present some collected data on the farmers behavior, that we used to tune the system and the models during our tests, and can be useful for future system deployments.

Reduction of the problem instance

Considering the NP-hardness of LF-STPP and LF-SP, we discuss a hierarchical approach, namely the Split Algorithm, which follows the *divide and conquer* paradigm by breaking down the input problem, when possible, into smaller sub-problems. These sub-problems may be individually solved by any model to build an optimal solution.

Algorithm 8: Split Algorithm

input : vertex set V, candidate lead farmers $\mathcal{L}$, opt. model ξ, par^*
output : An assignment of lead farmers and tasks. ;

solution $= \emptyset$;
$G = (V, \emptyset)$;
for $i, j \in V$ **do**
  if $\xi =$ *trajectory opt* **then**
    for $f \in \mathcal{L}, f \neq j$ **do**
      $w_0 = \omega^f_{f,i}$ if $f \neq i$ else 0;
      if $w_0 + \omega^f_{i,j} + \omega^f_{j,f} \leq b_f$ **then**
        $G.add\_edge((i, j))$;
  else
    if $i \in \mathcal{L} \wedge \omega_{i,j} \leq b_i$ **then**
      $G.add\_edge((i, j))$;
$\mathcal{G} = connected\_components(G)$;
for $G' \in \mathcal{G}$ **do**
  farmers $\mathcal{F} = V(G')$;
  solution = solution $\cup$ $\xi(\mathcal{F}, \mathcal{L}, par^*)$
return *solution*

The algorithm considers the vertex set $V = \mathcal{F}$, and builds a graph representing the problem instance by considering only edges that represent feasible tasks: i.e., the edge (i, j) is considered only if a candidate lead farmer can travel and visit the farmer j, starting from the location of farmer i, according to the required workload and farmer availability. The algorithm then computes the connected components of graph G, which represent the simpler sub-problems; solves them by using the same optimization model of the original problem, but applied to the single connected components; finally, it computes the global solution by merging the solutions of the sub-problems. Notice that, the notation $par*$ in the input represents the set of all parameters defining the optimization model (e.g., the selection cost h_i or the assignment costs c_ij).
The split phase is polynomial in the number of farmers ($|\mathcal{F}|$), while the optimization model for the reduced instances (connected components found by the Split Algorithm) may require an exponential number of iterations. By construction, no tasks may

exist between two components (i.e., farmers are too far away), and therefore the sub-problems can be solved independently leading to the optimal solution. In Theorem 5.3.1 we formally state this property and the algorithm optimality.

Theorem 5.3.1. *The solution of the Split Algorithm applied to the optimization model LF-STPP [LF-SP] is optimal.*

Proof. Let us assume that the solution s found by the Split Algorithm is not optimal for the original problem LF-STPP [LF-SP]. This means there exists an optimal solution s^* of the original problem LF-STPP [LF-SP] which utilizes assignment edges that are not included in any unique connected component of the graph constructed by the Split Algorithm. This implies that in s^* there must be a lead farmer f belonging to a component C, which is given the task to visit some farmer i in a different connected component C'. Nevertheless, this is only possible if the visit of f to i requires a workload lower than b_f. In details, for the problem LF-STPP, it must be $\omega^f_{f,i} + \omega^f_{i,f} \leq b_f$ [for the problem LF-SP, it must be $\omega_{f,i} \leq b_f$]. Nevertheless if this were true, the Split Algorithm would have included f and i in the same connected component, which contradicts the initial hypothesis. □

A Greedy Algorithm

Despite the simplifications introduced with the LF-SP formulation and the hierarchical approach provided by the Split Algorithm, the proposed problems are still NP-hard and cannot be solved in reasonable time if the problem instance is particularly large.

For this reason, we propose a greedy approach for the LF-SP, which runs in polynomial time, to be adopted for large problem instances. The greedy approach has very small computational requirements and, as we experimentally show in Section 5.3.4, performs close to the optimal solution in all the considered operational settings of our experiments.

The Greedy Algorithm iteratively assigns smartphones (i.e., selects lead-farmers), and related tasks, in a greedy manner. It starts by selecting the farmers having lower

Algorithm 9: Greedy Lead-Farmer Deployment

input : farmers $\mathcal{F}$, candidate lead farmers $\mathcal{L}$, par^*
output : Lead farmers $\mathcal{L}_{sol}$ and tasks $\mathcal{T}_{sol}$ assignment.

$\mathcal{L}_{sol} = \emptyset$, $\mathcal{T}_{sol} = \emptyset$;
while $|\mathcal{F}| > 0$ **do**
  $\Lambda = \{(f, t, R) \mid f \in \mathcal{L} : f \notin \mathcal{L}_{sol}, (t, R) \in sub\_routine(f, \mathcal{F})\}$;
  $(f^*, t^*, R^*) = \arg\min_{(f,t,R)\in\Lambda}(h_f + R)/|t|$;
  $\mathcal{L}_{sol} = \mathcal{L}_{sol} \cup f^*$;
  $\mathcal{T}_{sol} = \mathcal{T}_{sol} \cup t^*$;
  $\mathcal{F} = \{j \mid j \in \mathcal{F}\ s.t.\ \nexists (f, j) \in t^* \wedge j \neq f^*\}$;
for $(i, j) \in \mathcal{T}_{sol}$ **do**
  if $j \in \mathcal{L}_{sol}$ **then** $\mathcal{T}_{sol} = \mathcal{T}_{sol} \setminus (i, j)$;
return $\mathcal{L}_{sol}$, $\mathcal{T}_{sol}$

Algorithm 10: Greedy *sub_routine*

input : farmer f, un-assigned farmers $\mathcal{F}$, $par*$
output : An assignment of tasks t and its cost c, for the farmers f.

$t = \emptyset$; //tasks ;
$\omega_t = 0$; //tasks workload ;
$R = 0$; //tasks obj cost ;
while $\exists j \in \mathcal{F}$ *s.t.* $\omega_{f,j} \leq b_f \wedge \omega_{f,j} + \omega_t \leq M_f$ **do**
  $\mathcal{F} = \{j \mid \omega_{f,j} \leq b_f \wedge \omega_{f,j} + \omega_t \leq M_f\}$ $j^* = \arg\min_{j \in \mathcal{F}} c_{fj}$;
  $\mathcal{F} = \mathcal{F} \setminus j^*$;
  $t = t \cup \{(f, j*)\}$;
  $R = R + c_{fj}$;
return t, R

cost, calculated as in Equation 5.8, divided by the number of basic-farms connected. The algorithm selects new lead-farmers until all the basic farmers are covered. Finally, it iteratively removes redundant visits to lead-farmers. The algorithm is therefore polynomial in the number of farmers $|\mathcal{F}|$.

5.3.3 Crowd-sensing framework - A Real-Field Application

In this section we describe and evaluate the proposed framework in a real testbed in Kenya. We also provide useful real-field data for parameter tuning of the deployment models.

Busia testbed

We developed our testbed in Kenya, in the Busia region, at the beginning of 2018. We deployed the mobile crowd-sensing framework to test the mobile application, collect preliminary results and data, and validate the applicability of the proposed approach in a real case study.

We handed out 27 smartphones to the local charities in Busia. On their turn, the charities gave the smartphones to lead farmers selected on the sole basis of local people recognition and elections.

Figure 5.16 shows a snapshot of the crowd-sensing framework after 9 months of operation. The white circles with a red cross represent the lead farmer positions while the small blank points are sensing operations performed by lead farmers. When the point is green, or when a blue or violet area is around the blank points, a disease was detected. These points prove that the deployed framework is actually able to collect data around the region: thanks to the incentives described in Section 5.3.1, lead farmers have been actively participating in the system since their election, and have been continuously collecting measurements in their neighborhood.

Figure 5.17 shows two neighbor Cassava fields: the crop on the left is owned by a lead farmer while the one on the right belongs to a non collaborative farmer, which is not visited by any of the lead farmers. The figure shows how the two farms, which are adjacent to each other, present huge differences in productivity: while the

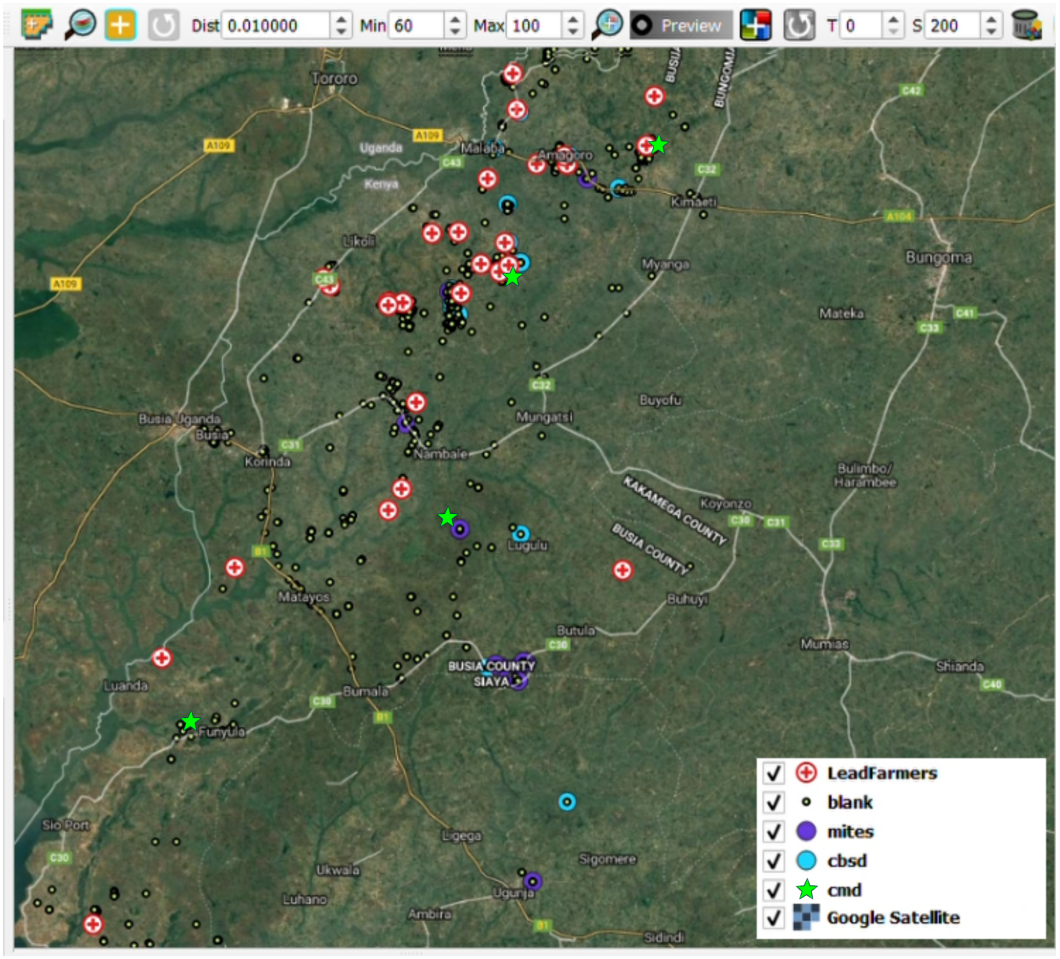


Figure 5.16. Snapshot of crowd-sensing framework after 9 months of operation.

lead farmer is able to visibly increase the field production, the other farmer still has poor yield.

In fact, the lead farmer, who had access to disease related notification, was able to detect infected plants and remove them. In the long run, the lead farmer eventually understood which plants are healthy for new seeding. It is worth to mention that, we show two close fields instead of the same field before and after the framework deployment, because there are many time-dependent factors (e.g. the number of rainy and sunny days) which can influence the crops and bias the evaluation.

Deployment data

Figure 5.16 shows how a deployment performed by charities, without any specific indications, results ineffective: some areas have high lead-farmer density (i.e. at the top of the map) while others have too few lead farmers to be able to completely carry on the sensing tasks in their proximity (i.e. at the bottom left of the map).

Thus, through the analysis of the performance of the existing crowd-sensing framework and by speaking with farmers, we collected the necessary information to

Figure 5.17. Cassava crop w/ and w/o framework.

perform our problem parameter tuning, and optimize the selection of lead farmers and their task allocation.

Table 5.3, shows the data collected considering the behavior of lead farmers in terms of time devoted to the framework, availability to move from farm to farm, and mobility characteristics. To collect these data, we considered 24 lead farmers in our initial test-bed (according to the charity, non-optimal configuration). The 24 farmers periodically visited their neighbor farms to sense data according to the mobile crowd-sensing model. The table reports the average, median and maximum traveled distance for each visit and the number of distinct visited neighbors farmers, per month, in 9 months of analysis. In summary, the average distance travelled by lead farmers to visit nearby farms is about $2km$ and they usually visit around 6 neighbors per month. The overall average speed for farmers (removing car and bus trips) is about $3m/s$, the inspection time is typically around 1 hours.

It is worth to note that, these measures are still dependent on the current deployment: for example a lead farmer can have only one close neighbor at a distance of $5km$ while another can have several neighbors in $100m$.

We use the average values of all the above measures to set our model parameters for an optimized lead farmer selection and task assignment (subset of farms assigned to a lead farmer for monitoring).

Considering the collected data, we assume a monthly workload for each lead farmer with at least one visit for each neighbor. We empirically evaluated that about 1 visit to each farm, per month, is sufficient to ensure the efficacy of the intervention. Therefore we roughly set k_v to 1/ month. We evaluated that lead farmers accept to work at most 4 hours for a single trip, including the inspection time, i.e. set to an average of 1 hour per visit. We set the maximum workload to 15 hours for month.

Table 5.3. Lead Farmers Data

Lead Farmer	Distance to basic farmers			Visited farmers
	avg (m)	Median (m)	Max (m)	
Farmer 1	603.96	527.81	1,145.80	12
Farmer 2	4,193.47	4,111.48	4,759.88	3
Farmer 3	422.69	530.02	612.68	7
Farmer 4	225.40	225.40	121.71	2
Farmer 5	235.41	265.52	337.90	5
Farmer 6	822.92	407.58	2,055.84	17
Farmer 7	290.26	290.26	309.58	2
Farmer 8	256.43	242.58	314.46	5
Farmer 9	397.73	407.52	449.66	4
Farmer 10	5,514.52	5,514.52	5,718.29	2
Farmer 11	714.07	818.26	1,115.99	8
Farmer 12	365.18	426.46	482.13	4
Farmer 13	1,862.52	454.93	4,146.51	5
Farmer 14	536.80	670.85	1,137.00	7
Farmer 15	729.33	729.33	792.92	2
Farmer 16	166.884	95.2305	437	4
Farmer 17	232.67	237.50	372.96	9
Farmer 18	220.86	220.86	267.40	2
Farmer 19	1345.75	1389.5	2514.85	6
Farmer 20	763.12	763.12	1,333.59	2
Farmer 21	1,850.12	1,422.83	7,173.16	13
Farmer 22	395.19	405.62	645.96	9
Farmer 23	1,282.38	1,282.38	2,433.65	2
Farmer 24	312.66	301.32	466.82	9
Total	**989**	**905**	**1631**	**6**

Finally, we consider a lead farmer deployment cost of 100$, which is just the price of the Nuru smartphone, i.e. Tecno Camon CA6S [192].

5.3.4 Deployment Models - Performance Evaluation

We now evaluate the proposed deployment solutions in two different settings: simulations and real field experiments. Simulations have the goal of investigating the performance of deployment models under a wide spectrum of operative settings, while experiments are performed in a real case scenario, constituted by our test-bed in Kenya. In both settings we use the Gurobi Optimizer [30] to solve the deployment problem.
For the purpose of showing the device deployments that are obtained through the

different proposed models, we initially consider a small scenario, with 23 farms in a small area (see Fig. 5.22). In particular, Figure 5.19 shows that the solution of LF-STPP (5.1) requires fewer lead farmers (namely, only 3 lead farmers) than the solution of LF-SP (5.2), shown in Figure 5.20 (which instead selects 4 lead farmers). Figure 5.21 shows the greedy approach which, in the same setting, selects 5 lead farmers. We recall that the approach provided by LF-STPP lets lead farmers perform multiple visits in a single trip. This allows the lead farmers to spare some of their time. When the constraints on traversed distance and daily workload allow so, the lead farmers can use this time to perform multiple visits in their trips. Hence the solutions of LF-STPP uses lead farmer time more efficiently than the approach of LF-SP.

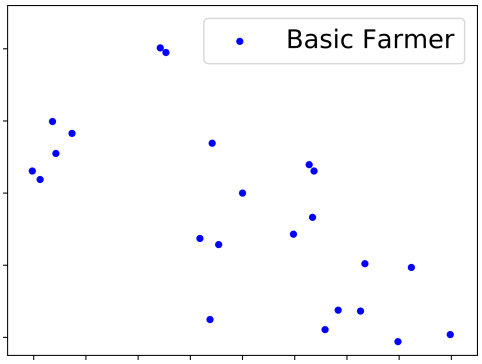


Figure 5.18. Initial AoI.

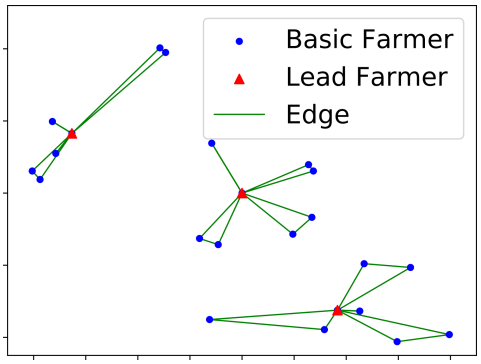


Figure 5.19. LF-STPP model solution.

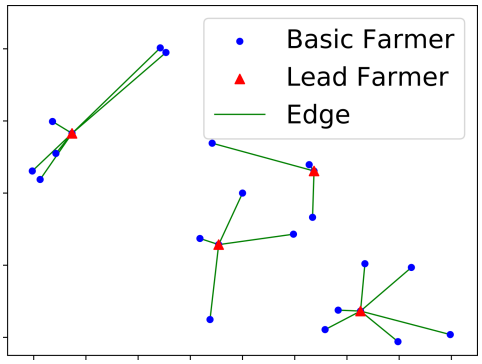


Figure 5.20. LF-SP model solution.

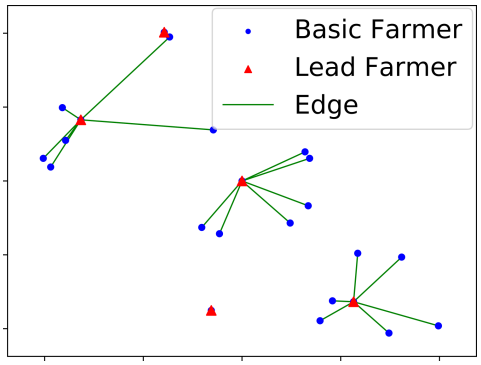


Figure 5.21. Greedy Algorithm solution.

Figure 5.22. Examples of deployment models.

Simulations

In simulation scenarios, where not otherwise stated, we consider a large area, i.e., $140km^2$, with several random uniformly distributed farms. We use a *Random* deployment model as a benchmark, which is meant to reflect the currently adopted approach, where lead farmers are chosen by elections or because of their prestige in

the community, with no correlation to their position in the region, nor any attention to the actual efficiency of their role in the system.

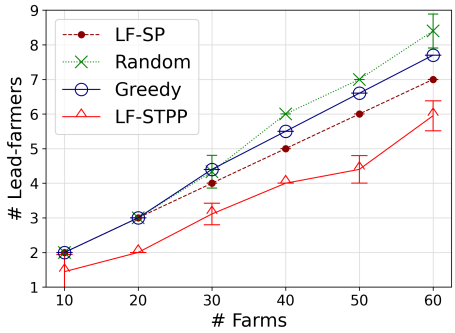


Figure 5.23. Number of selected lead-farmers at varying number of farms

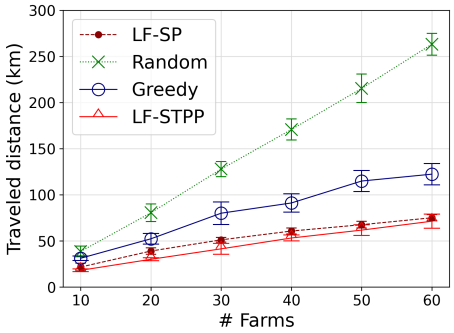


Figure 5.24. Traveled distance by lead-farmers at varying number of farms

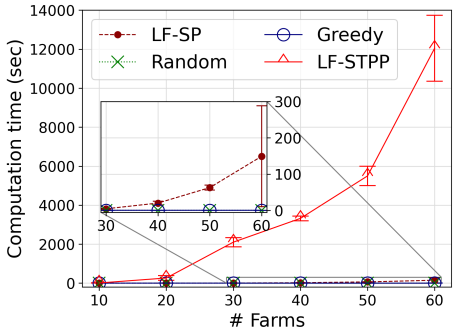


Figure 5.25. Computation times (seconds)

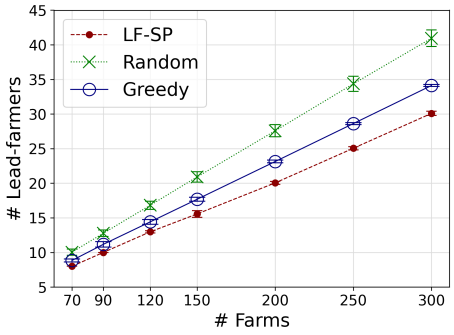


Figure 5.26. Number of selected lead-farmers at varying number of farms

We set the farmer constraints, and the visit parameters, according to the data collected in the real-field in Busia (Section 5.3.3), in a monthly setting. We run around 30 simulations for each case, and the error bars indicate the standard deviation of the plotted values.

Figure 5.23 shows the number of lead-farmers employed by the models by increasing the number of farms. The LF-STPP solution has the lowest number of lead farmers, which is always around 10% of the farmers in the area, while the LF-SP solution uses around 15% more lead-farmers. Finally, the *Greedy* solution, despite its simplicity, has performance close to the LF-SP optimal, while all the algorithms perform better than the *Random* deployment. Notice that, the *Random* performance diverges by increasing the complexity of the scenario, which is also evident in Figure 5.26, where we consider a larger scale experiment.

The performance of the framework is further investigated in Figure 5.24, which shows the total distance traveled by the farmers. If the crowd-sensing system provides rewards for the tasks, this metrics may reflect the cost for each solution. The LF-STPP has still the best performance, also in terms of travelled distance, despite the fact that it uses fewer lead farmers. This is mostly due to a better design of the

lead farmers' routes for LF-STPP.
The improvement of the LF-STPP is at the expense of a higher computation time. We underline that for large problem instances, computation time may become prohibitive, thus justifying the resort to a simplified model such as the LF-SP or to the Greedy Algorithm. Figure 5.25 shows the significant computation time (in seconds) required by LF-STPP, even with only 60 farms, with respect to other solutions.
The computation time of this plot is calculated executing the Split Algorithm approach, which significantly reduces the processing time with respect to the corresponding original problem formulation (i.e., between 2x to 10x in our simulations). In general, the improvement is more than proportional with respect to the number of sub-problems of the Split Algorithm, as the required computation time increases exponentially with the size of the problems.

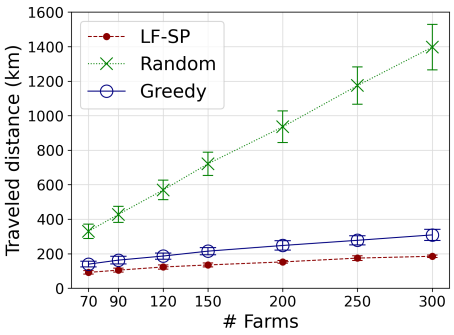


Figure 5.27. Traveled distance by lead-farmers at varying number of farms

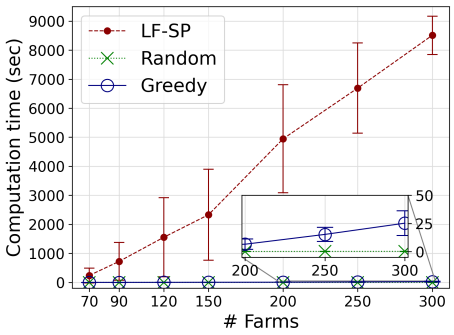


Figure 5.28. Computation times (seconds)

We now investigate the performance of the proposed solutions for larger problem settings, with up to 300 farms. Due to the huge computation time of LF-STPP we do not consider it in these scenarios. Figure 5.26 shows the required lead farmers when varying the number of farms. Results confirm the algorithm performance trends: LF-SP has the best performance, followed by the *Greedy* and by the *Random* deployment model. In particular, the figure shows how the performance diverges when the number of farms grows. Figure 5.27 shows a similar rank among the algorithms also when the considered performance metric is the total distance traveled by lead farmers. Despite the larger number of farmers used by the Greedy and Random approach, the LF-SP approach ensures a lower traversed distance thanks to a smarter association of lead farmers to basic ones.
Finally, Figure 5.28 shows the computation time of the three solutions. The LF-SP model has the highest computation time, with around 2.5 hours of computation for 300 farms, while the other solutions have negligible computation time. In conclusion, our set of simulations demonstrates that LF-SP is the most efficient deployment model. The *Greedy* model also has a good performance (close to the LF-SP solution), with a much lower computation time, which make it a promising solution for very large problem settings.

Heuristic Comparison

We extend our performance studies to larger problem settings, considering up to 900 farms. Considering these settings, we evaluate our heuristic in comparison with solutions to similar problems in the domain of mobile device deployment. We recall that our problem shows unique characteristics, being it related to humans actively using the deployed devices, and adopted in developing country scenarios, with poor network infrastructure, and poor road conditions. However, we did our best to position our work in the broader context of device deployment studies. We consider a similar deployment problem for mobile robots, namely the mobile sensor sweep coverage problem, where a given number of targets must be monitored by mobile robots to periodically collect data. By logically mapping lead farmers to mobile robots, and farms to targets, we compare our work to CSWEEP [195], a mobile robot deployment algorithm that minimizes the number of robots needed to achieve the required monitoring frequency of targets (i.e., their sweep period). Other related works, also mentioned in Section 5.3.5, show some similarities to ours, but address different objectives and are not directly comparable.

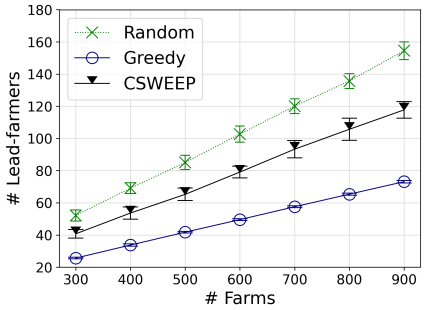


Figure 5.29. Number of selected lead-farmers at varying number of farms.

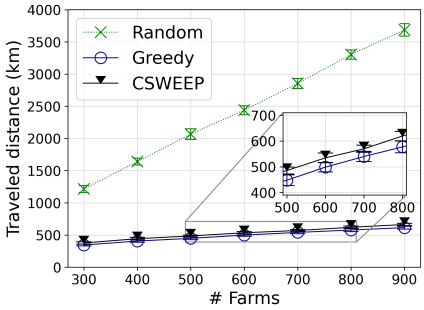


Figure 5.30. Traveled distance by lead-farmers at varying number of farms.

CSWEEP may be used in our application context to guide decisions related to lead farmer election and their task assignments. It computes a set of tours by partitioning a global tour including all the targets. We configure CSWEEP to take account of the heterogeneous constraints and requirements of the different farmers, and we let it partition the global tour accordingly.

Figures 5.29 and 5.30 show the algorithm performance. Notice that, due to the considerable computation time of LF-STPP and LF-SP we do not propose their application in a large scenarios although they perform better than the greedy approach both in terms of number of deployed devices and of traversed distance.

Figure 5.29 shows the number of required lead farmers, under a varying number of farms. The results show that *Greedy* always has the best performance. In particular, with 900 farmers *Greedy* uses only about 70 smartphones while CSWEEP requires 120 smartphones ($\sim 70\%$ more than *Greedy*), and *Random* needs 155 smartphones ($\sim 120\%$ more than *Greedy*). The figure also shows how the performance diverges when the number of farms grows.

Figure 5.30 also shows the superiority of *Greedy* with respect to CSWEEP in terms of total distance traversed by lead farmers. In particular, the figure shows that although CSWEEP requires a higher number of lead farmers than *Greedy*, the total traversed distance under CSWEEP is higher than with *Greedy* mostly due to an inefficient global tour partitioning and task assignment, which does not allow to use the full potential of the lead farmers.

Real-field experiments

Figure 5.35 considers a comparison among the discussed approaches when applied to our real test-bed scenario, in the Busia region, in Kenya. For this scenario we compare LF-STPP, LF-SP and Greedy, to the original device deployment provided by the charity management, based on local lead farmer elections. The experiments show how the two optimization models can effectively reduce the number of necessary smartphone devices while providing the same or better coverage and load balancing among lead farmers. The initial device deployment performed according to the charity management, and the related task assignment, are shown in Figure 5.31. The charity deployment requires 27 smartphones, at a cost of 100 each. Figure 5.32 shows the deployment and task assignment optimization, according to LF-STPP. Such a solution reduces the number of devices from 27 to 14, with an overall reduction of 48%, while achieving a more uniform coverage and meeting the workload constraints imposed by the farmers. Figure 5.33 shows the optimization according to the solution to the simplified model LF-SP. The model is able to actually reduce the number of needed smartphones from 27 to 17, with a cost reduction of around 37% with respect to the charity approach, showing a small cost increase with respect to LF-STPP. Finally, Figure 5.34 shows the deployment according to the Greedy Algorithm, which performs close to LF-SP. In this case, the number of needed smartphone decreases to 18, with 33% of reduction in cost.

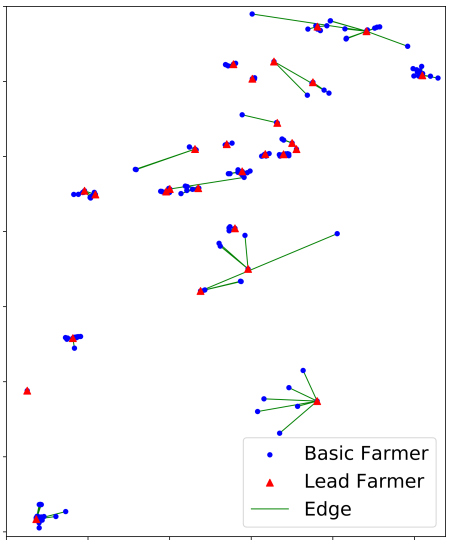


Figure 5.31. Previous deployment.

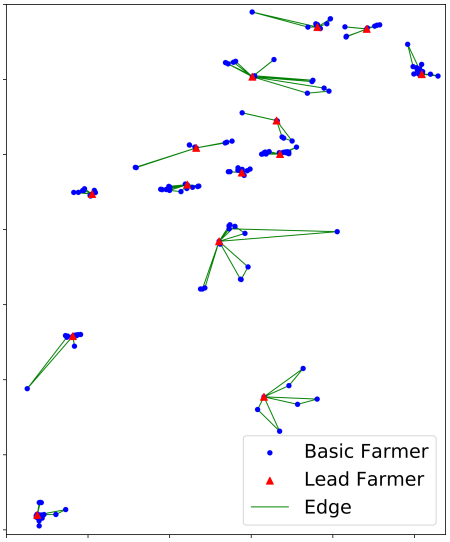


Figure 5.32. LF-STPP model solution.

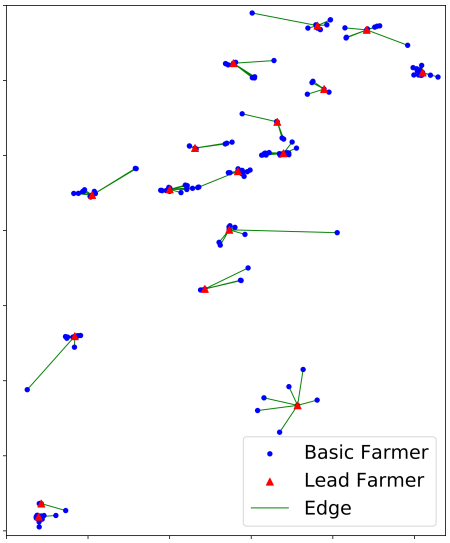


Figure 5.33. LF-SP model solution.

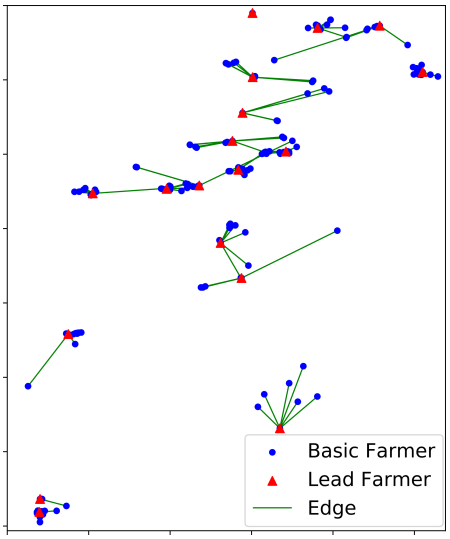


Figure 5.34. Greedy Algorithm solution.

Figure 5.35. Busia region lead-farmer assignment.

It is worth to mention that the optimization proposed for the test-bed implementation, namely the LF-STPP solution, significantly reduces deployment costs with respect to a best-effort, election based, device deployment scheme. Despite the lower number of sensing devices being used, we observed an improvement in crop production due to the more uniform deployment of the framework and a better load balancing among the lead farmers. While the improvement brought by the optimization models is moderate in an area where the crowd-sensing framework was already in use before the re-configuration, we underline that such a re-configuration permits the use of the otherwise unused extra devices in larger areas, extending the application of the framework to currently uncovered regions.

We also underline that cost reduction is very critical for deploying the described crowd-sensing framework in new regions where the smartphones must be financed by government entities of by charity associations.

In fact, in developing countries, with poor funding availability, it is fundamental to be able to extend the use of the framework to entire regions, minimizing the cost per region, and maximizing the extent of the served area. Finally, it is worth noting that the models give a reliable and unbiased tool for selecting the most suitable lead farmer selection and monitoring task assignment, which is relevant to ensure uniform service coverage and quality over the area of interest.

5.3.5 Related Work

Mobile crowd-sensing frameworks With the rising of IoT, smartphones and mobile sensors, several works have been devoted to mobile crowd-sensing frameworks in different domains [196,197]. Nevertheless, in agriculture and food domain only few works have been proposed, despite the high number of challenges due to increasing demand of food and climate changes.

J. Mohite et al. proposed a *Rural Participatory Sensing* (RuPS) framework for crop monitoring, in which farmers collect data about plant diseases [198]. The proposed crowd-sensing framework presents a reward mechanism to stimulate participation, and uses collected data, which are validated by experts, to analyze possible epidemics and eventually providing alerts to the farmers. In a subsequent work, P. Singh et al. [199] introduce a crowd-sensing system to provide consulting and help farmers in diagnosing plant diseases with the aid of experts. The system, using data collected from the farmers, helps experts to reduce possible diseases, with a limited number of interactions with the farmers for providing help. Unlike ours, these works do not provide direct feedback to farmers as we do through the Nuru application. In the cited works, the proposed smartphone applications are not able to immediately identify diseases offline, but they can only collect data or ask for expert advice. As they cannot work offline, and rely on the continuous availability of expert advisors, the proposed approaches have limited chances of applicability in developing countries, due to the poor internet connectivity, and limited infrastructure available.

A first solution for scenarios with scarce connectivity is proposed by P. Gupta et al. in [200]. They propose a *Rural Agriculture Participatory Sensing* with delay tolerant networks to allow seamless stream of data from farmers to the experts and improve crops health. They especially address developing countries in which network connectivity is extremely poor. They propose a multi-hop communication where a farmer acts as relay node to provide and extend device communication range. However, the farmers are often too far from each other to connect in a multi-hop network. Finally, in [201] Z. Jiajin et al. propose a social network sensor platform to provide precision agriculture and several services to farmers. Sensors are placed in the crop to collect relevant data (e.g. temperature, humidity and so on). The farmers using the smartphones and the Bluetooth technology can recover data from sensors and participate in the sensing framework.

Incentive Mechanisms As mobile crowd-sensing systems often include incentives mechanisms to stimulate users participation [183, 184, 202] and improve the quality of collected data [202–204], we now discuss some related proposals.
X. Fang et al. propose two incentive mechanisms based on rewards [202]. The first one provides a fixed reward (shared by all the users), while the second one offer a payment to users providing service. These approaches can be adopted to pay and encourage lead farmers to visit their neighborhood, if required.
To improve sensing and data quality, H. Jin et al. study Quality of Information (QoI) aware incentive mechanisms [204]. As QoI is highly related to poor sensor quality, noise, or lack in sensor calibration, different users may have different data quality/resolution, asking for proportional rewards. They propose a system that jointly considers different prices (incentives), and QoI of the user, to collect quality data at a reasonable price. Also H. Jin et al. propose a novel incentive mechanism, named Theseus, which improves data and sensing quality [205]. In this case, the proposed system offers payments to incentivize high-effort sensing from workers. As in real field the sensed data may be unreliable and noisy, the participants can improve data quality by increasing their effort on sensing (e.g., using more often the phone on sunny days/hours). A more sophisticated mechanism is proposed by X.

Tian et al. in [203]. The authors propose an online pricing mechanism to purchase high-quality data, coming from participants in an online manner. The proposed mechanism determines how much the system should pay for new data, considering both the limited budget and the expected quality level. The authors also investigate budget minimization for a fixed required quality level.

The aforementioned solutions help mobile crowd-sensing framework to collect data with higher quality and encourage participants. If needed (e.g., with heterogeneous private devices, or in case of no collaborative farmers) they can be integrated in our framework. Notice that, the deployment models have already the possibility to specify participant costs and participant-to-task costs.

Deployment Models The device deployment problem is common to many application fields, where mobile devices enable a service over an area of interest, and each device executes different tasks in a geographically distributed manner. We refer the reader to the work in [206] for a survey on task assignment in crowd sensing frameworks. There is also a large body of work addressing the device deployment and task assignment for mobile sensors and mobile robots [18, 37, 195, 207]. Some of these works can be adapted to our scenario with proper modifications meant to tackle the specific characteristics of crowd sensing in developing countries, such as specific patterns of human mobility, heterogeneity in the definition of people constraints, lack of infrastructure, and need to minimize costs. Few works consider the problem of patrolling a set of target points under visit frequency constraints. Gao et al. in [207] consider the sweep coverage problem, where mobile sensors are deployed to periodically cover a set of targets, in a given region. They find a set of disjoint tours for each mobile sensor to continuously monitor the targets. While this approach can be used to assign smartphones and tasks to lead farmers, it considers a different objective function, and does not aim at minimizing the number of mobile monitoring devices (i.e., smartphones handed over to lead farmers) as we do in our work.

A closer problem was studied in the work by Li et al. [195] which minimizes the number of mobile sensors to cover all the targets within their visit frequency constraints (i.e., the sweep period). In section 5.3.4 we adopt this approach for our performance evaluation. In order to have fair comparisons, we consider a variant of this approach which takes account of the heterogeneous and limited farmer capabilities.

We note that, although we did our best to seek the most related problems in the literature, our problem is novel and presents some unique characteristics which require a specifically suited solution. We recall that, despite the existing works for mobile sensors and mobile robots show some resemblance to ours, our deployment model considers human participants and their interactions. In our scenario, mobility is only partially controllable, mostly in an indirect manner, through incentives; users are highly heterogeneous, in term of speed, time availability, capabilities and also mutual interactions. The same infrastructure, with poor network connectivity and road conditions, poses limitations to users' movements which are uncommon in other domains. In conclusion, despite some similarities with problems defined in

other domains, the unique challenge tackled in this work requires ad-hoc problem formulations and related solutions.

5.3.6 Conclusions and Future Work

In this work we show how smart technologies and tools may support agriculture and the production of high quality food, especially in developing countries. In fact, in these countries, the poor availability of skilled personnel (plant pathologists, agronomists) and the lack of supporting infrastructure (road, internet connectivity) make crop fields particularly vulnerable to the outbreak of plant diseases.

We propose a crowd-sensing framework, in which we hand out smartphones to some lead farmers, who are in charge of providing the fundamental sensing activities of the framework. Smartphones are endowed with the Nuru application, which detects plant diseases at early stages, and constitutes the key component of the crowd sensing framework, when introduced at large scale in the farmer population. As devices are typically provided in a limited amount by government agencies or by charity institutions, to make the system operational we address the device deployment problem, according to which a limited number of cheap smartphones is handed-out for free to farmers both as a mobile sensing device, and as an incentive to encourage participation. We propose two analytical models to minimize the number of required smartphones while providing sufficient geographical coverage. We also study a greedy algorithm which trades off efficiency and scalability. We evaluate the proposed solutions through simulations and real experiments (e.g., a testbed in the Busia region in Kenya), showing that the proposed solutions outperform the previously adopted ones in terms of monitoring accuracy, coverage completeness and homogeneity, with lower cost, almost halving the number of required monitoring devices.

In a future work, we envision the use of networks of drones to substitute smartphones, towards an autonomous sensing framework. Drones have the key advantage of wireless networking and mobility in the same device, which makes them ideal to move fast between farms and offload data. Moreover, drones require little to no human control, while smartphones are traveled by a farmers which can partially cooperate and reach only close farms. We expect a massive improvement in the performance. In terms of cost (a single drone is expected to replace more than 30-40 smartphones) and in terms of performance (a drone can visit several farms in few hours and prioritize critical areas).

Chapter 6

Conclusion

In this thesis we studied applications and related solutions for networks of aerial drones, to bring them closer to the adoption in real scenarios. To this end we explicitly considered the unique characteristics of drones, such as the limited battery lifetime and communication capabilities, in the design of our solutions.

In chapter 2 we addressed the problem of trajectory planning in safety-critical scenarios and we proposed several solutions that take in account the drones' physical constraints, i.e., limited energy and capabilities. We introduced a multi-trip MILP formulation to optimize the trajectories, considering battery recharging/replacement and data offloading at the depot. We provided a constant factor approximation algorithm as well as a novel genetic-based algorithm. By means of extensive simulations we demonstrated how our solutions outperform existing state-of-art work. Finally, we studied the algorithms' applicability in a real test-bed.

In chapter 3 we exploited the communication capabilities of drones in the design of path planning algorithms. We first considered the connected early-target inspection problem: drones must monitor a set of targets while being connected to the depot to offload the collected data. We demonstrated the performance of our solution in providing fresh data to the base station with the respect to existing solution, in both simulations and real-field experiments. Then, we considered the communication capabilities as means for the drones to share their local observations of the environment, and coordinate during the exploration. We considered an unknown environment and we proposed a virtual-force based approach to let drones autonomously inspect the area of interest under *uncertainty*. Through simulations and real-field experiments we validated our proposal and we show that it discovers new events $30 - 40\%$ faster than existing algorithms.

In chapter 4 we discussed the challenges of guaranteeing stable and reliable multi-hop communication during the exploration of an area of interest, due to the fast and unconstrained mobility of drones. Therefore, we developed a new solution for packet routing in the challenging domain of FANETs. We exploited the device controllable mobility to facilitate network routing, and we demonstrated superior performance to existing routing algorithms.

Finally, we investigate three novel applications of UAVs networks, in the context of post-disaster management, parcel delivery and food safety and security.

6.1 Acknlowledgements

The presented work has been partially supported by the North Atlantic Treaty Organization (NATO), under the Science for Peace and Security grants: "Hybrid Sensor Networks for Emergency Critical Scenarios" number SPS G4936; and "SeaSec: DroNets for Maritime Border and Port Security" number SPC G5828.

Bibliography

[1] M. Erdelj, E. Natalizio, K. R. Chowdhury, and I. F. Akyildiz, "Help from the sky: Leveraging uavs for disaster management," *IEEE Pervasive Computing*, vol. 16, no. 1, pp. 24–32, 2017.

[2] J. Cohen, "Wakemed health & hospitals joins forces with ups, faa for drone pilot," 2019. [Online]. Available: https://www.modernhealthcare.com/care-delivery/wakemed-health-hospitals-joins-forces-ups-faa-drone-pilot

[3] FAO, "E-Agriculture in action: drones for agriculture," 2018. [Online]. Available: http://www.fao.org/3/I8494EN/i8494en.pdf

[4] J. C. Hodgson, R. Mott, S. M. Baylis, T. T. Pham, S. Wotherspoon, A. D. Kilpatrick, R. Raja Segaran, I. Reid, A. Terauds, and L. P. Koh, "Drones count wildlife more accurately and precisely than humans," *Methods in Ecology and Evolution*, vol. 9, no. 5, pp. 1160–1167, 2018.

[5] J. K. Stolaroff, C. Samaras, E. R. O'Neill, A. Lubers, A. S. Mitchell, and D. Ceperley, "Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery," *Nature communications*, vol. 9, no. 1, pp. 1–13, 2018.

[6] P. Cohn, A. Green, M. Langstaff, and M. Roller, "Commercial drones are here: The future of unmanned aerial systems," *McKinsey & Company*, 2017.

[7] L. Gupta, R. Jain, and G. Vaszkun, "Survey of important issues in uav communication networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, 2015.

[8] V. Hassija, V. Chamola, A. Agrawal, A. Goyal, N. C. Luong, D. Niyato, F. R. Yu, and M. Guizani, "Fast, reliable, and secure drone communication: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2021.

[9] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, and M. Debbah, "A tutorial on uavs for wireless networks: Applications, challenges, and open problems," *IEEE communications surveys & tutorials*, vol. 21, no. 3, pp. 2334–2360, 2019.

[10] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-uav systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3340–3385, 2019.

[11] A. Baltaci, E. Dinc, M. Ozger, A. Alabbasi, C. Cavdar, and D. Schupke, "A survey of wireless networks for future aerial communications (facom)," *IEEE Communications Surveys & Tutorials*, 2021.

[12] D. S. Lakew, U. Sa'ad, N.-N. Dao, W. Na, and S. Cho, "Routing in flying ad hoc networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[13] H. Wang, H. Zhao, J. Zhang, D. Ma, J. Li, and J. Wei, "Survey on unmanned aerial vehicle networks: A cyber physical system perspective," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1027–1070, 2019.

[14] Horizon Europe Work Programm, "2021-2022 civilsecurity for society," 2021. [Online]. Available: https://ec.europa.eu/info/funding-tenders/opportunities/docs/2021-2027/horizon/wp-call/2021-2022/wp-6-civil-security-for-society_horizon-2021-2022_en.pdf

[15] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[16] P. Oberlin, S. Rathinam, and S. Darbha, "A transformation for a heterogeneous, multiple depot, multiple traveling salesman problem," in *IEEE American Control Conference*, 2009.

[17] D. Kim, L. Xue, D. Li, Y. Zhu, W. Wang, and A. O. Tokuta, "On theoretical trajectory planning of multiple drones to minimize latency in search-and-reconnaissance operations," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3156–3166, Nov 2017.

[18] N. Bartolini, A. Coletta, and G. Maselli, "On task assignment for early target inspection in squads of aerial drones," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 2123–2133.

[19] N. Bartolini, A. Coletta, G. Maselli, and A. Khalifeh, "A multi-trip task assignment for early target inspection in squads of aerial drones," *IEEE Transactions on Mobile Computing*, 2020.

[20] N. Bartolini, A. Coletta, G. Maselli, M. Piva, and D. Silvestri, "Genpath-a genetic multi-round path planning algorithm for aerial vehicles," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–6.

[21] N. Bartolini, A. Coletta, M. Prata, and C. Serino, "On connected deployment of delay-critical fanets," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 9720–9727.

[22] K. Miranda, A. Molinaro, and T. Razafindralambo, "A survey on rapidly deployable solutions for post-disaster networks," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 117–123, 2016.

[23] N. Bartolini, A. Coletta, and G. Maselli, "Side: Self driving drones embrace uncertainty," *IEEE Transactions on Mobile Computing*, 2021.

[24] N. Bartolini, A. Coletta, A. Gennaro, G. Maselli, and M. Prata, "Mad for fanets: Movement assisted delivery for flying ad-hoc networks," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 315–325.

[25] A. Coletta, G. Maselli, M. Piva, and D. Silvestri, "Danger: a drones aided network for guiding emergency and rescue operations," in *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2020, pp. 303–304.

[26] N. Bartolini, A. Coletta, G. Maselli, and M. Piva, "Druber: a trustable decentralized drone-based delivery system," in *Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, 2020.

[27] FAO, "The future of food and agriculture – alternative pathways to 2050." 2018. [Online]. Available: www.fao.org/3/I8429EN/i8429en.pdf

[28] N. Bartolini, A. Coletta, G. Maselli, and D. Hughes, "On optimal crowd-sensing task management in developing countries," in *IEEE PerFlow '20 (PerCom)*, 2020.

[29] A. Coletta, N. Bartolini, G. Maselli, A. Kehs, P. McCloskey, and D. P. Hughes, "Optimal deployment in crowd sensing for plant disease diagnosis in developing countries," *IEEE Internet of Things Journal*, 2020.

[30] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[32] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[33] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[34] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[36] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[37] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*, P. Toth and D. Vigo, Eds. SIAM, 2014, no. 18.

[38] J. Fakcharoenphol, C. Harrelson, and S. Rao, "The k-traveling repairman problem," in *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ser. SODA '03. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, pp. 655–664.

[39] C. S. Martin and M. R. Salavatipour, "Minimizing latency of capacitated k-tours," *Algorithmica*, vol. 80, no. 8, pp. 2492–2511, Aug 2018. [Online]. Available: https://doi.org/10.1007/s00453-017-0337-x

[40] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, pp. 1–10, 2011.

[41] T. K. Ralphs, L. Kopman, W. R. Pulleyblank, and L. E. Trotter, "On the capacitated vehicle routing problem," *Mathematical programming*, vol. 94, no. 2-3, pp. 343–359, 2003.

[42] F. Afghah, A. Razi, J. Chakareski, and J. Ashdown, "Wildfire monitoring in remote areas using autonomous unmanned aerial vehicles," *IEEE MiSARN*, 2019.

[43] S. Lee and J. R. Morrison, "Decision support scheduling for maritime search and rescue planning with a system of uavs and fuel service stations," in *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, 2015.

[44] Q. Wu, Y. Zeng, and R. Zhang, "Joint trajectory and communication design for multi-uav enabled wireless networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 2109–2121, 2018.

[45] O. Zorlu, "Routing unmanned aerial vehicles as adapting to capacitated vehicle routing problem with genetic algorithms," in *IEEE 7th International Conference on Recent Advances in Space Technologies (RAST)*, 2015.

[46] H. Binol, E. Bulut, K. Akkaya, and I. Guvenc, "Time optimal multi-uav path planning for gathering its data from roadside units," *IEEE VTC-Fall*, pp. 1–5, 01 2018.

[47] M. H. Dominguez, S. Nesmachnow, and J.-I. Hernández-Vega, "Planning a drone fleet using artificial intelligence for search and rescue missions," in *IEEE 24th International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*. IEEE, 2017, pp. 1–4.

[48] J. Faigl and P. Váňa, "Unsupervised learning for surveillance planning with team of aerial vehicles," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, 2017.

[49] S. Islam and A. Razi, "A path planning algorithm for collective monitoring using autonomous drones," in *53rd IEEE Conference on Information Sciences and Systems (CISS)*, 2019.

[50] T. Setter and M. Egerstedt, "Energy-constrained coordination of multi-robot teams," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1257–1263, 2016.

[51] M.-I. Popescu, H. Rivano, and O. Simonin, "Multi-robot patrolling in wireless sensor networks using bounded cycle coverage," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2016, pp. 169–176.

[52] A. Yazici, G. Kirlik, O. Parlaktuna, and A. Sipahioglu, "A dynamic path planning approach for multirobot sensor-based coverage considering energy constraints," *IEEE Transactions on Cybernetics*, vol. 44, no. 3, pp. 305–314, 2013.

[53] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "Deployment of mobile robots with energy and timing constraints," *IEEE Transactions on robotics*, vol. 22, no. 3, pp. 507–522, 2006.

[54] DJI, "Agras t16," 2019. [Online]. Available: https://www.dji.com/it/t16

[55] ——, "Flame wheel arf kit F550," 2019. [Online]. Available: https://www.dji.com/it/flame-wheel-arf/spec

[56] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *J. ACM*, vol. 7, no. 4, pp. 326–329, Oct. 1960.

[57] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[58] A. Fotouhi, M. Ding, and M. Hassan, "Understanding autonomous drone maneuverability for internet of things applications," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2017, pp. 1–6.

[59] M. B. Milam, R. Franz, and R. Murray, "A new computational approach to real-time trajectory generation for constrained mechanical systems," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2000.

[60] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[61] I. Davidson and S. S. Ravi, "Agglomerative hierarchical clustering with constraints: Theoretical and empirical results," in *Knowledge Discovery in Databases (KDD)*, A. M. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 59–70.

[62] F. R. Bach and M. I. Jordan, "Learning spectral clustering," in *Advances in neural information processing systems*, 2004, pp. 305–312.

[63] K. Goss, R. Musmeci, and S. Silvestri, "Realistic models for characterizing the performance of unmanned aerial vehicles," in *26th International Conference on Computer Communication and Networks, ICCCN*. Vancouver, BC, Canada, July 31 - Aug. 3, 2017, pp. 1–9.

[64] J. Lee, *A First Course in Combinatorial Optimization*. Cambridge University Press, 2004.

[65] E. Lawler, *Combinatorial optimization - networks and matroids*. New York: Holt, Rinehart and Winston, 1976.

[66] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a submodular set function subject to a matroid constraint," in *IPCO*, June 2007.

[67] M. Fisher, G. Nemhauser, and L. Wolsey, "An analysis of approximations for maximizing submodular set functions – II," *Math. Prog. Study*, vol. 8, pp. 73–87, 1978.

[68] Ilya Krutov, "Lenovo system x3550 m5 (machine type 8869)," 2019. [Online]. Available: https://lenovopress.com/lp0067-lenovo-system-x3550-m5-machine-type-8869

[69] DJI, "Naza-m v2 compact designed multirotor autopilot system," 2019. [Online]. Available: https://www.dji.com/it/naza-m-v2

[70] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6753–6760.

[71] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[72] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[73] E. Yanmaz and H. Guclu, "Stationary and mobile target detection using mobile wireless sensor networks," in *INFOCOM IEEE Conference on Computer Communications Workshops*. IEEE, 2010, pp. 1–5.

[74] J. Hu, H. Zhang, L. Song, R. Schober, and H. V. Poor, "Cooperative internet of uavs: Distributed trajectory design by multi-agent deep reinforcement learning," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6807–6821, 2020.

[75] F. Arvin, T. Krajník, A. E. Turgut, and S. Yue, "Cosϕ: Artificial pheromone system for robotic swarms research," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 407–412.

[76] A. Dutta, A. Ghosh, and O. P. Kreidl, "Multi-robot informative path planning with continuous connectivity constraints," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3245–3251.

[77] J. Banfi, A. Q. Li, I. Rekleitis, F. Amigoni, and N. Basilico, "Strategies for coordinated multirobot exploration with recurrent connectivity constraints," *Autonomous Robots*, vol. 42, no. 4, pp. 875–894, 2018.

[78] J. Banfi, A. Q. Li, N. Basilico, I. Rekleitis, and F. Amigoni, "Multirobot online construction of communication maps," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2577–2583.

[79] J. Reich, V. Misra, D. Rubenstein, and G. Zussman, "Connectivity maintenance in mobile wireless networks via constrained mobility," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 5, pp. 935–950, 2012.

[80] F. Schiano, A. Franchi, D. Zelazo, and P. R. Giordano, "A rigidity-based decentralized bearing formation controller for groups of quadrotor uavs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 5099–5106.

[81] F. Schiano and P. R. Giordano, "Bearing rigidity maintenance for formations of quadrotor uavs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1467–1474.

[82] Y. Yang, D. Constantinescu, and Y. Shi, "Connectivity-preserving swarm teleoperation with a tree network," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 3624–3629.

[83] J. Scherer and B. Rinner, "Multi-robot persistent surveillance with connectivity constraints," *IEEE Access*, vol. 8, pp. 15 093–15 109, 2020.

[84] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, "Multi-uav exploration with limited communication and battery," in *IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 2230–2235.

[85] G. A. Hollinger and S. Singh, "Multirobot coordination with periodic connectivity: Theory and experiments," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 967–973, 2012.

[86] S. Hayat, E. Yanmaz, C. Bettstetter, and T. X. Brown, "Multi-objective drone path planning for search and rescue with quality-of-service requirements," *Autonomous Robots*, vol. 44, no. 7, pp. 1183–1198, 2020.

[87] N. Goddemeier, K. Daniel, and C. Wietfeld, "Role-based connectivity management with realistic air-to-ground channels for cooperative uavs," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 5, pp. 951–963, 2012.

[88] K. Goss, R. Musmeci, and S. Silvestri, "Realistic models for characterizing the performance of unmanned aerial vehicles," in *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.

[89] E. Kantor and D. Peleg, "Approximate hierarchical facility location and applications to the shallow steiner tree and range assignment problems," in *Italian Conference on Algorithms and Complexity*. Springer, 2006, pp. 211–222.

[90] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems: revised reprint*. SIAM, 2012.

[91] M. Mahdian, Y. Ye, and J. Zhang, "Improved approximation algorithms for metric facility location problems," in *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 2002, pp. 229–242.

[92] DJI, "Phantom 4 pro v2.0," 2021. [Online]. Available: https://www.dji.com/it/phantom-4-pro-v2

[93] R. P. FOUNDATION, "Raspberry pi zero w," 2021. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[94] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, "Better approach to mobile ad-hoc networking (batman)," *IETF draft*, 2008.

[95] A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch, "Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey," *Networks*, vol. 72, no. 4, pp. 411–458, 2018.

[96] M. S. Gerber, "Predicting crime using twitter and kernel density estimation," *Decision Support Systems*, vol. 61, pp. 115–125, 2014.

[97] J. Shahrabi and R. Pelot, "Kernel density analysis of maritime fishing traffic and incidents in canadian atlantic waters," 2009.

[98] P. A. Apostolopoulos, G. Fragkos, E. E. Tsiropoulou, and S. Papavassiliou, "Data offloading in uav-assisted multi-access edge computing systems under resource uncertainty," *IEEE Transactions on Mobile Computing*, 2021.

[99] E. Arribas, V. Mancuso, and V. Cholvi, "Coverage optimization with a dynamic network of drone relays," *IEEE Transactions on Mobile Computing*, vol. 19, no. 10, pp. 2278–2298, 2019.

[100] E. Natalizio, N. Zema, E. Yanmaz, L. D. P. Pugliese, and F. Guerriero, "Take the field from your smartphone: Leveraging uavs for event filming," *IEEE Trans. Mob. Comp.*, 2019.

[101] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Mobile unmanned aerial vehicles (uavs) for energy-efficient internet of things communications," *IEEE Transactions on Wireless Communications*, vol. 16, no. 11, pp. 7574–7589, 2017.

[102] A. Bujari, C. E. Palazzi, and D. Ronzani, "A comparison of stateless position-based packet routing algorithms for fanets," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, 2018.

[103] P. S. Bithas, V. Nikolaidis, A. G. Kanatas, and G. K. Karagiannidis, "Uav-to-ground communications: Channel modeling and uav selection," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 5135–5144, 2020.

[104] W. Khawaja, I. Guvenc, D. W. Matolak, U.-C. Fiebig, and N. Schneckenburger, "A survey of air-to-ground propagation channel modeling for unmanned aerial vehicles," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2361–2391, 2019.

[105] A. A. Khuwaja, Y. Chen, N. Zhao, M.-S. Alouini, and P. Dobbins, "A survey of channel modeling for uav communications," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2804–2821, 2018.

[106] T. S. Rappaport *et al.*, *Wireless communications: principles and practice*. prentice hall PTR New Jersey, 1996, vol. 2.

[107] C. H. Liu, X. Ma, X. Gao, and J. Tang, "Distributed energy-efficient multi-uav navigation for long-term communication coverage by deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1274–1285, 2019.

[108] N. Bartolini, G. Bongiovanni, T. La Porta, and S. Silvestri, "On the vulnerabilities of the virtual force approach to mobile sensor deployment," *IEEE Transactions on Mobile Computing*, vol. 13, no. 11, pp. 2592–2605, Nov. 2014.

[109] L. Lin and M. A. Goodrich, "Hierarchical heuristic search using a gaussian mixture model for uav coverage planning," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2532–2544, 2014.

[110] P. Sujit and D. Ghose, "Search using multiple uavs with flight time constraints," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 2, pp. 491–509, 2004.

[111] J. S. Bellingham, M. Tillerson, M. Alighanbari, and J. P. How, "Cooperative path planning for multiple uavs in dynamic and uncertain environments," in *Proceedings of the 41st IEEE Conference on Decision and Control.* IEEE, 2002, pp. 2816–2822.

[112] K. Shang, S. Karungaru, Z. Feng, L. Ke, and K. Terada, "Periodic re-optimization based dynamic branch and price algorithm for dynamic multi-uav path planning," in *IEEE International Conference on Mechatronics and Automation.* IEEE, 2013, pp. 581–586.

[113] K. Konolige, D. Fox, B. Limketkai, J. Ko, and B. Stewart, "Map merging for distributed robot navigation," in *Proceedings 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 1. IEEE, 2003, pp. 212–217.

[114] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006.

[115] T. Cieslewski, S. Lynen, M. Dymczyk, S. Magnenat, and R. Siegwart, "Map api-scalable decentralized map building for robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6241–6247.

[116] S. Golodetz, T. Cavallari, N. A. Lord, V. A. Prisacariu, D. W. Murray, and P. H. Torr, "Collaborative large-scale dense 3d reconstruction with online inter-agent pose optimisation," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 11, pp. 2895–2905, 2018.

[117] Y. Zhang, L. Chen, Z. XuanYuan, and W. Tian, "Three-dimensional cooperative mapping for connected and automated vehicles," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 8, pp. 6649–6658, 2019.

[118] Z. Zhu, W. Jiang, L. Yang, and Z. Luo, "Indoor multi-robot cooperative mapping based on geometric features," *IEEE Access*, vol. 9, pp. 74 574–74 588, 2021.

[119] I. Andersone, "The characteristics of the map merging methods: A survey." *Computer Science (1407-7493)*, vol. 43, 2010.

[120] D. Rodriguez-Losada, F. Matia, and A. Jimenez, "Local maps fusion for real time multirobot indoor simultaneous localization and mapping," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 2. IEEE, 2004, pp. 1308–1313.

[121] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[122] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, pp. 832–837, 1956.

[123] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a gps-denied environment," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1814–1820.

[124] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking*. ACM, 2004, pp. 45–57.

[125] K. Ali, H. X. Nguyen, Q.-T. Vien, P. Shah, and M. Raza, "Deployment of drone-based small cells for public safety communication system," *IEEE Systems Journal*, vol. 14, no. 2, pp. 2882–2891, 2020.

[126] M. Shahbazi, "Professional drone mapping," in *Unmanned Aerial Systems*. Elsevier, 2021, pp. 439–464.

[127] M. R. Garey, R. L. Graham, and D. S. Johnson, "Some np-complete geometric problems," in *Proceedings of the eighth annual ACM symposium on Theory of computing*, 1976, pp. 10–22.

[128] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, 1977. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397577900123

[129] Z. Xie and J. Yan, "Kernel density estimation of traffic accidents in a network space," *Computers, environment and urban systems*, vol. 32, no. 5, pp. 396–406, 2008.

[130] Z. I. Botev, J. F. Grotowski, D. P. Kroese *et al.*, "Kernel density estimation via diffusion," *The annals of Statistics*, vol. 38, no. 5, pp. 2916–2957, 2010.

[131] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[132] Y. Sun, D. Xu, D. W. K. Ng, L. Dai, and R. Schober, "Optimal 3d-trajectory design and resource allocation for solar-powered uav communication systems," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4281–4298, 2019.

[133] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline scheduling for real-time systems: EDF and related algorithms.* Springer Science & Business Media, 2012, vol. 460.

[134] DJI, "Matrice 600 pro," 2019. [Online]. Available: https://www.dji.com/it/matrice600-pro/info#specs

[135] R. Pi, "Raspberry pi 2 model b," 2020. [Online]. Available: https://www.raspberrypi.org

[136] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, *Mobile ad hoc networking.* John Wiley & Sons, 2004.

[137] J. Harri, F. Filali, and C. Bonnet, "Mobility models for vehicular ad hoc networks: a survey and taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, 2009.

[138] J. Liu, Q. Wang, C. He, K. Jaffrès-Runser, Y. Xu, Z. Li, and Y. Xu, "Qmr: Q-learning based multi-objective optimization routing protocol for flying ad hoc networks," *Computer Communications*, vol. 150, 2020.

[139] M. Asadpour, K. A. Hummel, D. Giustiniano, and S. Draskovic, "Route or carry: Motion-driven packet forwarding in micro aerial vehicle networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 3, pp. 843–856, 2016.

[140] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless networks*, vol. 7, no. 6, 2001.

[141] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized link state routing protocol (olsr)," 2003.

[142] J. Chroboczek, "The babel routing protocol," RFC 6126, April, Tech. Rep., 2011.

[143] D. Johnson, Y.-c. Hu, D. Maltz *et al.*, "The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4," RFC 4728, Tech. Rep., 2007.

[144] C. Perkins, E. Belding-Royer, and S. Das, "Rfc3561: Ad hoc on-demand distance vector (aodv) routing," 2003.

[145] K. Peters, A. Jabbar, E. K. Cetinkaya, and J. P. Sterbenz, "A geographical routing protocol for highly-dynamic aeronautical networks," in *IEEE Wireless Communications and Networking Conference*. IEEE, 2011.

[146] S. Liu, T. Fevens, and A. E. Abdallah, "Hybrid position-based routing algorithms for 3d mobile ad hoc networks," in *The 4th International Conference on Mobile Ad-hoc and Sensor Networks*. IEEE, 2008.

[147] A. E. Abdallah, T. Fevens, and J. Opatrny, "Randomized 3d position-based routing algorithms for ad-hoc networks," in *3rd Annual International Conference on Mobile and Ubiquitous Systems-Workshops*. IEEE, 2006.

[148] W.-S. Jung, J. Yim, and Y.-B. Ko, "Qgeo: Q-learning-based geographic ad hoc routing protocol for unmanned robotic networks," *IEEE Communications Letters*, vol. 21, no. 10, 2017.

[149] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012.

[150] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, USA, 1992.

[151] R. Elliot, "Ieee standard definitions of terms for antennas," *IEEE Transactions on Antennas and Propagation*, vol. 31, no. 6, pp. 1–29, 1983.

[152] Q. Huang, R. Huang, W. Hao, J. Tan, R. Fan, and Z. Huang, "Adaptive power system emergency control using deep reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 11, no. 2, 2019.

[153] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[154] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[155] A. Bujari, O. Gaggi, C. E. Palazzi, and D. Ronzani, "Would current ad-hoc routing protocols be adequate for the internet of vehicles? a comparative study," *IEEE Internet of Things Journal*, vol. 5, no. 5, 2018.

[156] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, 2019.

[157] S. H. Alsamhi, O. Ma, M. S. Ansari, and F. A. Almalki, "Survey on collaborative smart drones and internet of things for improving smartness of smart cities," *Ieee Access*, vol. 7, pp. 128 125–128 152, 2019.

[158] N. Strochlic, "Surprising ways drones are saving lives, National Geographic," 2017. [Online]. Available: https://www.nationalgeographic.com/magazine/2017/06/explore-drones-for-good/

[159] M. McNabb, "Drones for good: UNICEF is calling on drone operators for vaccine delivery. 2018. [Online] https://dronelife.com/2018/06/14/drones-for-good-unicef-is-calling-on-drone-operators-for-vaccine-delivery," 2018. [Online]. Available: dronelife.com/2018/06/14/drones-for-good-unicef-is-calling-on-drone-operators-for-vaccine-delivery/

[160] B. Mishra, D. Garg, P. Narang, and V. Mishra, "Drone-surveillance for search and rescue in natural disaster," *Computer Communications*, vol. 156, pp. 1–10, 2020.

[161] L. Ferranti, S. D'Oro, L. Bonati, E. Demirors, F. Cuomo, and T. Melodia, "Hiro-net: Self-organized robotic mesh networking for internet sharing in disaster scenarios," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9.

[162] Z. Lu, G. Cao, and T. La Porta, "Teamphone: Networking smartphones for disaster recovery," *IEEE Transactions on Mobile Computing*, vol. 16, no. 12, pp. 3554–3567, 2017.

[163] J. Clement, "Digital buyers worldwide 2021," 2019. [Online]. Available: https://www.statista.com/statistics/251666/number-of-digital-buyers-worldwide/

[164] A. P. Air, "Amazon," 2020. [Online]. Available: https://www.amazon.com/Amazon-Prime-Air/

[165] D. I. Insights, "Drone applications database," Jan 2019. [Online]. Available: https://www.droneii.com/project/drone-applications-database

[166] B. Popper, "Dji's new drone can fly in rain or snow," Feb 2017. [Online]. Available: https://www.theverge.com/2017/2/26/14701198/dji-drone-matrice-200-industrial-enterprise

[167] R. D'Andrea, "Guest editorial can drones deliver?" *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, 2014.

[168] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[169] FedEx, "Delivery cost," 2020. [Online]. Available: https://www.fedex.com/en-us/online/rating.html

[170] J. Elias, "Alphabet's wing launches first commercial drone delivery," 2019. [Online]. Available: https://www.cnbc.com/2019/10/18/alphabets-wing-launches-first-commercial-drone-delivery.html

[171] T. Dasu, Y. Kanza, and D. Srivastava, "Geofences in the sky: herding drones with blockchains and 5g," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018, pp. 73–76.

[172] S. B. H. Youssef, S. Rekhis, and N. Boudriga, "A blockchain based secure iot solution for the dam surveillance," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–6.

[173] I. Jensen, D. Selvaraj, and P. Ranganathan, "Blockchain technology for networked swarms of unmanned aerial vehicles (uavs)," in *IEEE 20th International WoWMoM Symposium 2019*.

[174] S. Aggarwal, M. Shojafar, N. Kumar, and M. Conti, "A new secure data dissemination model in internet of drones," in *IEEE International Conference on Communications (ICC) 2019*.

[175] P. Mehta, R. Gupta, and S. Tanwar, "Blockchain envisioned uav networks: Challenges, solutions, and comparisons," *Computer Communications*, vol. 151, pp. 518–538, 2020.

[176] United Nations, "Sustainable development goals, 17 goals to transform our world," 2019. [Online]. Available: https://www.un.org/sustainabledevelopment/

[177] M. Tsan, S. Totapally, M. Hailu, and B. K. Addom, *The Digitalisation of African Agriculture Report 2018–2019*. CTA, 2019.

[178] M. L. Blum and J. Szonyi, "Investment requirements in extension to achieve zero hunger and adapt to climate change," *Journal of Agricultural Science and Technology. A*, vol. 4, no. 7A, 2014.

[179] L. Li, "Application of the internet of thing in green agricultural products supply chain management," in *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, vol. 1. IEEE, 2011, pp. 1022–1025.

[180] M. Abbasi, M. H. Yaghmaee, and F. Rahnama, "Internet of things in agriculture: A survey," in *2019 3rd International Conference on Internet of Things and Applications (IoT)*. IEEE, 2019, pp. 1–12.

[181] PlantVillage, "Plantvillage," 2019. [Online]. Available: https://plantvillage.psu.edu/

[182] PV, "Plantvillage Nuru - android," 2019. [Online]. Available: play.google.com/store/apps/details?id=plantvillage.nuru

[183] L. G. Jaimes, I. J. Vergara-Laurens, and A. Raij, "A survey of incentive techniques for mobile crowd sensing," *IEEE Internet of Things Journal*, vol. 2, no. 5, pp. 370–380, 2015.

[184] X. Zhang, Z. Yang, W. Sun, Y. Liu, S. Tang, K. Xing, and X. Mao, "Incentives for mobile crowd sensing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 54–67, 2015.

[185] S. Savary, L. Willocquet, S. J. Pethybridge, P. Esker, N. McRoberts, and A. Nelson, "The global burden of pathogens and pests on major food crops," *Nature ecology & evolution*, vol. 3, no. 3, p. 430, 2019.

[186] E.-C. Oerke, "Crop losses to pests," *The Journal of Agricultural Science*, vol. 144, no. 1, pp. 31–43, 2006.

[187] TensorFlow, "Tensorflow detection model zoo." 2019. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[188] A. Ramcharan, P. McCloskey, K. Baranowski, N. Mbilinyi, L. Mrisho, M. Ndalahwa, J. Legg, and D. P. Hughes, "A mobile-based deep learning model for cassava disease diagnosis," *Frontiers in Plant Science*, vol. 10, p. 272, 2019.

[189] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpls.2016.01419

[190] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in plant science*, vol. 8, p. 1852, 2017.

[191] GSMARENA, "Samsung galaxy s5," 2019. [Online]. Available: https://www.gsmarena.com/samsung_galaxy_s5-6033.php

[192] ——, "Tecno camon cm," 2019. [Online]. Available: https://www.gsmarena.com/tecno_camon_cm-9443.php

[193] S. Franzel, E. Kiptot, and A. Degrande, "Farmer-to-farmer extension: A low-cost approach for promoting climate-smart agriculture," in *The Climate-Smart Agriculture Papers*. Springer, 2019, pp. 277–288.

[194] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, 1st ed. New York, NY, USA: Cambridge University Press, 2011.

[195] M. Li, W. Cheng, K. Liu, Y. He, X. Li, and X. Liao, "Sweep coverage with mobile sensors," *IEEE Transactions on Mobile Computing*, vol. 10, no. 11, pp. 1534–1545, 2011.

[196] A. Capponi, C. Fiandrino, B. Kantarci, L. Foschini, D. Kliazovich, and P. Bouvry, "A survey on mobile crowdsensing systems: Challenges, solutions and opportunities," *IEEE Communications Surveys & Tutorials*, 2019.

[197] W. Zamora, C. T. Calafate, J.-C. Cano, and P. Manzoni, "A survey on smartphone-based crowdsensing solutions," *Mobile Information Systems*, vol. 2016, 2016.

[198] J. Mohite, Y. Karale, P. Gupta, S. Kulkarni, B. Jagyasi, and A. Zape, "Rups: Rural participatory sensing with rewarding mechanisms for crop monitoring," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2015, pp. 378–383.

[199] P. Singh, B. Jagyasi, N. Rai, and S. Gharge, "Decision tree based mobile crowdsourcing for agriculture advisory system," in *2014 Annual IEEE India Conference (INDICON)*. IEEE, 2014, pp. 1–6.

[200] P. Gupta, B. Jagyasi, B. Panigrahi, H. K. Rath, S. Pappula, and A. Simha, "A revolutionary rural agricultural participatory sensing approach using delay tolerant networks," *arXiv preprint arXiv:1612.02927*, 2016.

[201] Z. Jiajin, C. Lichang, D. Qingsong, Z. Haidong, and Z. Yonghua, "A social networks integrated sensor platform for precision agriculture," in *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*. IEEE, 2014, pp. 131–136.

[202] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012, pp. 173–184.

[203] X. Tian, W. Zhang, Y. Yang, X. Wu, Y. Peng, and X. Wang, "Toward a quality-aware online pricing mechanism for crowdsensed wireless fingerprints," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 5953–5964, 2018.

[204] H. Jin, L. Su, D. Chen, K. Nahrstedt, and J. Xu, "Quality of information aware incentive mechanisms for mobile crowd sensing systems," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2015, pp. 167–176.

[205] H. Jin, B. He, L. Su, K. Nahrstedt, and X. Wang, "Data-driven pricing for sensing effort elicitation in mobile crowd sensing systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2208–2221, 2019.

[206] J. Wang, L. Wang, Y. Wang, D. Zhang, and L. Kong, "Task allocation in mobile crowd sensing: State-of-the-art and future opportunities," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3747–3757, 2018.

[207] X. Gao, J. Fan, F. Wu, and G. Chen, "Approximation algorithms for sweep coverage problem with multiple mobile sensors," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 990–1003, 2018.