Optimization of UAVs deployment and coordination for exploration and monitoring applications

Igor Dias da Silva

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Optimisation du déploiement et de la coordination de drones pour les applications d'exploration et de surveillance

Igor Dias da Silva

COATI, Inria, I3S, CNRS, Université Côte d'Azur

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : David COUDERT, Directeur de Recherche, Centre Inria d'Université Côte d'Azur

Co-encadrée par : Christelle CAILLOUET, Maîtresse de Conférences, Université Côte d'Azur

Soutenue le : 21/09/2023

Devant le jury, composé de :
Emmanuel CHAPUT, Professeur, INP Toulouse
Enrico NATALIZIO, Professeur, UL, Nancy
Hervé RIVANO, Professeur, INSA, Lyon
Yann BUSNEL, Professeur, IMT Nord Europe, Lille

OPTIMISATION DU DÉPLOIEMENT ET DE LA COORDINATION DE DRONES POUR LES APPLICATIONS D'EXPLORATION ET DE SURVEILLANCE

Optimisation of UAVs deployment and coordination for exploration and monitoring applications

Igor Dias da Silva

Jury :

Rapporteurs

Emmanuel CHAPUT, Professeur, INP Toulouse
Enrico NATALIZIO, Professeur, UL, Nancy

Examinateurs

Hervé RIVANO, Professeur, INSA, Lyon
Yann BUSNEL, Professeur, IMT Nord Europe, Lille

Directeur de thèse

David COUDERT, Directeur de Recherche, Centre Inria d'Université Côte d'Azur

Co-encadrante de thèse

Christelle CAILLOUET, Maîtresse de Conférences, Université Côte d'Azur

---

Université Côte d'Azur

Igor Dias da Silva

Optimisation du déploiement et de la coordination de drones pour les applications d'exploration et de surveillance

xiii+112 p.

Ce document a été préparé avec LaTeX2e et la classe `these-ISSS` version v. 2.10.

Impression : main.tex – 22/9/2023 – 17:53

Révision pour la classe : these-ISSS.cls,v 2.10 2020/06/24 14:16:37 mpelleau

To all who have supported me during this journey.

Optimisation du déploiement et de la coordination de drones pour les applications d'exploration et de surveillance

Résumé

Les véhicules aériens sans pilote (UAV) ont suscité une attention considérable récemment. Ces appareils volants autonomes présentent plusieurs avantages en termes de coûts de déploiement et de capacités de traitement, ce qui en fait une solution prometteuse pour un large éventail d'applications militaires et commerciales. Dans cette thèse, nous avons étudié deux applications : la création d'un réseau aérien pour connecter des capteurs mobiles à une station de base et la recharge sans fil de capteurs fixes au sol. Dans la première application, nous voulons communiquer avec des capteurs mobiles dans un scénario de catastrophe sans infrastructure de communication préexistante. Notre objectif dans cette application est de déterminer les meilleurs plans de vol pour les UAVs. Les UAVs doivent maintenir un chemin connecté entre la station de base et les capteurs pour permettre aux capteurs d'envoyer des informations critiques si nécessaire. Nous optimisons la distance totale parcourue par les UAVs et l'énergie consommée. La deuxième application concerne un ensemble de capteurs au sol fixes que nous devons recharger. La technologie de récupération d'énergie, basée sur l'échange de signaux radio-fréquence, permettant de recharger les batteries des capteurs sans fil. Nous avons proposé une solution en 2 étapes où d'abord, un programme linéaire détermine où chaque drone doit aller et combien de temps il doit y rester pour s'assurer que les capteurs reçoivent suffisamment d'énergie. Ensuite, des algorithmes gloutons d'ordonnancement déterminent l'ordre dans lequel les UAVs visitent ces points, en veillant à ce qu'ils ne se gênent pas mutuellement ou ne réduisent pas l'efficacité de la charge. Nous minimisons le temps total nécessaire pour recharger tous les capteurs dans les deux étapes.

Mots-clés : UAVs, drones, optimisation, coordination, programmation linéaire, algorithmes

Optimisation of UAVs deployment and coordination for exploration and monitoring applications

Abstract

Unmanned Aerial Vehicles (UAVs) have gained tremendous attention lately. These autonomous flying devices have several advantages in terms of deployment cost and processing capabilities, making them a promising solution for a wide range of military and commercial applications. In this thesis, we have studied two applications: creating a flying network to connect mobile sensors to a base station and wirelessly charging fixed ground sensors. In the first application, we want to communicate with mobile sensors in a disaster scenario without pre-existing communication infrastructure. Our goal in this application is to determine the optimal flight plans for the UAVs. The UAVs must maintain a connected path between the base station and the sensors to allow the sensors to send critical information when necessary. We optimise the UAVs' total distance travelled and energy consumed. The second application has a set of fixed ground sensors we need to recharge. Equipping the UAVs and sensors with power harvesting technology allows us to recharge these sensors wirelessly. We proposed a 2 step solution where first, a linear program finds where each drone needs to go and how long it should stay there to ensure the sensors get enough power. Then, greedy scheduling algorithms determine the order in which UAVs visit these spots, ensuring they do not get in each other's way or reduce the charging efficiency. We minimise the total time to charge all sensors in both steps.

Keywords: UAVs, drones, optimisation, coordination, linear programming, algorithms

Remerciements

First, I would like to thank my supervisors, Christelle and David, for their guidance and support during this journey. You were incredibly patient and supportive during some hard times, and I will always be grateful. I learned a lot from you and wish I could have learned more. It was a privilege to work with you.

Then I would like to thank my Family and Ianca for their support and love. It was tough to be away from you, but you always gave me the energy to keep going and reminded me that I would never be truly alone. I would also like to thank my friends, my second family. The ones from COATI, Inria, UBINET and random encounters in Nice. You kept me sane and made me feel at home.

Without any of you, I would not be here today. Each word of encouragement, each hug, and each laugh helped me more than you can know. I learned from all of you and grew with all of you. I hope to be able to give back all that you gave me.

Finally, I thank the members of the jury, Emmanuel Chaput, Enrico Natalizio, Hervé Rivano and Yann Busnel, for their time and attention in reviewing this thesis.

Thank you, merci, obrigado.

Contents

Annexes

Chapter 1

Introduction

Nowadays, we run across excellent online videos showing beautiful and unique places we want to visit. Many of these videos are recorded by unmanned aerial vehicles (UAVs). These drones used to be bigger and much more expensive but are becoming more and more accessible as technology advances. At present, they are our toys and flying cameras. However, these are not the only uses we have for them.

1.1 Types of drones

We usually associate the term "drone" with aerial vehicles, but by definition, a drone is any unmanned vehicle, including ground and underwater vehicles. In this thesis, we focus on aerial drones. There are many types of drones in different shapes and sizes. Some examples are shown in Figure 1.1. We can separate them into different categories depending on size, wing formation, weight, flying altitude, range, autonomy, propulsion system, and application.

(a) Multi-Rotor drone. (b) Fixed-Wing drone. (c) Different shapes.

Figure 1.1: Some different types of Drones.

Drones can be classified based on wing types, with two primary categories being multi-rotor and fixed-wing drones. Multi-rotor drones, depicted in Figure 1.1(a), are commonly encountered and instantly recognisable as drones. They are popular due to their affordability, ease of use, exceptional stability, and manoeuvrability. These drones can effortlessly take off, land vertically, maintain a stable hover, and quickly navigate tight spaces. While their flight time and payload capacity are usually limited, they can vary greatly. Their flight range can vary from 20 m to 10 km, just as their price ranges from €20 to €50 000.

On the other hand, there are fixed-wing drones, illustrated in Figure 1.1(b). They closely resemble aeroplanes in design and function. They rely on a propulsion system to move forward while the wings generate the necessary lift for sustained flight. Consequently, these drones require

adequate space for takeoff and landing but are more energy-efficient than their multi-rotor counterparts. They can reach higher speeds and have a more significant flight range. However, fixed-wing drones tend to be more expensive. While professional multi-rotor drones start at €900, professional fixed-wing drones start at €10 000. Additional variations exist that we do not see as often, such as single-rotor drones resembling helicopters and hybrid drones that combine characteristics of both multi-rotor and fixed-wing drones.

When considering weight, drones span a broad spectrum, with tiny ones weighing less than $0.2\ \mathrm{kg}$ and large-scale models exceeding $20\,000\ \mathrm{kg}$. The weight of a drone depends on its specific configuration, including the presence of cameras, sensors, weapons, and other equipment, which can significantly impact its overall weight. Some applications require large drones that can carry heavy payloads, while others benefit from small drones that are more agile and manoeuvrable. And often, using multiple small drones over a single large drone can be advantageous. However, designing systems that coordinate the actions of various drones is challenging.

1.2 Technologies

A commercial drone usually has a controller that uses radio frequency signals to send the pilot's commands. Some manufacturers promise a communication range of up to $15\mathrm{km}$ with video streaming for an average professional drone, while a toy drone usually flies up to $50\mathrm{m}$ away from the controller before losing connection. You can find the toy drone for €20 while the professional one is about €1600. Most controllers work in the $2.4\ \mathrm{GHz}$ to $5.8\ \mathrm{GHz}$ range and use technologies proprietary to drone manufacturers. But these proprietary technologies are not the only options for communication.

Jawhar, Mohamed, Al-Jaroodi, Agrawal, and Zhang (2017) mention several technologies we can use to communicate with drones. They show that with Wi-Fi, or more specifically IEEE 802.11a/b/g/n, we can obtain a communication range between 120 and $250\ \mathrm{m}$ and up to $150\mathrm{Mbps}$ of data rate. We can improve that by using WiMAX (IEEE 802.16), which reaches up to $56\mathrm{km}$ of communication range working in the 2 to $66\ \mathrm{GHz}$ band. With WiMAX, the data rate varies from 2 to $75\ \mathrm{Mbps}$.

We cannot always send the signal directly to the drone, so we can use a pre-existing network infrastructure to forward our commands instead. Cellular networks are a popular option, where the drone's communication range is unlimited as long as you and the drone can access the network. The problem with this approach is the increased latency in communication. Still, many studies with 5G and 6G networks focus on reducing latency and increasing the data rate, which are the central promises of these technologies (Mishra & Natalizio, 2020; Geraci et al., 2022). For example, Zolanvari, Jain, and Salman (2019) mention that with 5G we could reach $1\ \mathrm{Gbps}$ with only $1\ \mathrm{ms}$ delay in a perfect scenario, but any implementations will be constrained by the reality of available resources which is often far from the ideal plan.

While 5G and 6G are great solutions to communicate with our drones, we still have not reached the point where the connection is ubiquitous. Even in an area with network coverage, we might have problems if the UAV flies too high since the antennas are usually designed to communicate with users on the ground, not in the air. There are many applications in military and research where drones operate beyond the radio frequency line of sight in remote locations where cellular networks are unavailable or not an option. In these cases, satellite networks are an alternative (Li, Fei, & Zhang, 2019). They have much broader coverage, but the data rate is lower, and the delay

is higher. Some companies promise a data rate of $24\ \mathrm{Mbps}$ with a $132.5\ \mathrm{ms}$ delay while others promise a lower delay of $40\ \mathrm{ms}$ but only $128\ \mathrm{Kbps}$ (Zolanvari et al., 2019).

The advertised communication ranges and data rates tend to be overestimated compared to reality. You might achieve these communication ranges in a remote area without obstacles, but they will be much lower in a city. If you check the specifications of the €1600 professional drone, the announced streaming range of $15\mathrm{km}$ is actually between 9 and $15\mathrm{km}$ in a remote environment with almost perfect conditions. They do inform you eventually that in an urban environment, the expected range is actually between 1.5 and $3\mathrm{km}$. And this is not the only catch. These ranges are possible with FCC certification (Federal Communications Commission *FCC Starts Rulemaking on Licensed Spectrum for Unmanned Aircraft Use* (2023)), which allows you to use the full power of the radio. However, in Europe, you should follow the CE certification (Conformité Européenne *Commission Delegated Regulation (EU) 2019/945 of 12 March 2019 on Unmanned Aircraft Systems and on Third-Country Operators of Unmanned Aircraft Systems* (2019)). Using the CE certification, the range in perfect conditions drops to a maximum of $8\mathrm{km}$ and the range expected in an urban environment is not specified. The data rate is also lower in reality due to the interference of other devices and the environment, so the streamed video quality can also suffer. When raining, you should keep your drone inside.

But drones are not entirely dependent on the connection with a controller. Most drones have a built-in safety feature that makes them return to the take-off point when the connection is lost. This feature is possible thanks to the GPS module that allows drones to execute autonomous flights (Kwak & Sung, 2018). Drones can fly autonomously, even without GPS, by relying on their cameras, sensors and mapping techniques to estimate their position. This is not a trivial task, and many researchers are working on improving the methods for autonomous flights to make them more reliable and efficient (Balamurugan, Valarmathi, & Naidu, 2016; Tang et al., 2019).

Besides taking pictures, drones can collect data from their surroundings using various sensors. They can be equipped with thermal cameras, radar, ground penetrating radar, lasers, microphones and more. On top of that, drones can have speakers and lights to interact with their surroundings, and some can grab and drop things.

Overall, the specifications of drones vary significantly depending on the technologies used. More processing power means bulkier hardware and higher energy consumption. You can have it all, but it will cost you. Moreover, we can use a single drone to execute a task or multiple drones collaborating toward their goals. So for each application, it is essential to define the requirements well and choose suitable drones.

1.3 General Applications

All these technologies make drones incredibly versatile tools. They are suitable for a wide range of applications and are expected to be even more integrated into our daily lives in the future (Shakhatreh et al., 2019). This section presents some of the most common applications of drones.

Smart cities

Drones have emerged as valuable assets in civil and commercial applications within cities, offering various services, including surveillance and security. As mobile cameras, drones can effectively monitor buildings and crowds, aiding authorities with crowd management on public streets and at

large events. Due to their mobility, drones can capture images from various angles, providing valuable insights into crowd dynamics and events unfolding within them (Tang et al., 2021). Hence, they facilitate the identification and comprehension of emergencies and the tracking of suspects.

Drones can also collect traffic information to help us understand traffic patterns and monitor urban congestion (Khan, Ectors, Bellemans, Janssens, & Wets, 2017). They can even gather data on weather conditions and pollution levels across the city. This data can contribute to more effective urban planning and resource allocation.

Another significant application of drones in smart cities is package delivery. Numerous studies have explored the development of efficient drone-based delivery systems (Raivi, Huda, Alam, & Moh, 2023). Butterworth-Hayes (2019) show a list of 27 countries with drone delivery operations that range from food to medical utilities. This innovative approach to delivery offers increased speed, reduced costs, and improved accessibility in urban areas.

Furthermore, drones are great tools in construction and industrial sites. They can transport materials and tools, monitor worker activity, conduct remote infrastructure inspections, and gather valuable site data. By leveraging drones in these environments, site management becomes more efficient, secure, cost-effective, and scalable (Nikolic et al., 2013; Aiello, Hopps, Santisi, & Venticinque, 2020).

Considering the growing demand for enhanced connectivity in the 5G/B5G era, drones present a promising solution to improve users' Quality of Experience (Mishra & Natalizio, 2020). We can integrate drones with the communication infrastructure to expand coverage and increase data rates. As the users' distribution and needs change, these drones can change position allowing the network to adapt to new scenarios. This integration can be crucial in bridging connectivity gaps and ensuring a seamless user experience in urban areas. Just as we can use the cellular network to improve our communication with drones, we can use drones to improve our communication with the cellular network.

Remote environments, smart farms and forests

In smart farms, the use of drones offers a multitude of benefits. These unmanned aerial vehicles can gather essential data on soil conditions, plant growth, and overall plant health, enabling farmers to make informed decisions and promptly address any issues that may arise. By leveraging drone technology, farmers can optimise resource allocation, reduce costs, and improve crop yields, leading to both economic and environmental benefits. As early as 2007, Japan already used unmanned helicopters (UAVs) to spray over 10% of their rice plantations (Otto, Agatz, Campbell, Golden, & Pesch, 2018).

Furthermore, drones are an efficient solution for animal tracking on smart farms and in vast forested areas. These aerial devices streamline the laborious process of locating and monitoring animals, providing researchers and conservationists with valuable data. Using drones, researchers can gather accurate information on animal behaviour, migration patterns, and population dynamics (Torresan et al., 2017). This knowledge supports wildlife conservation efforts and facilitates the implementation of effective management strategies. Even in the ocean, aerial drones can obtain critical information for researchers to monitor wildlife, track the impact of pollution and manage disasters (Yang et al., 2022).

Drones are indispensable for surveillance and combating illegal activities in challenging environments like dense forests. Authorities employ drones to monitor vast areas and detect illegal deforestation, mining operations, and other environmentally destructive practices (Ota et al., 2019;

Giang et al., 2020). Data collected by drones help governments and organisations identify affected areas promptly and take appropriate measures to prevent further damage. Therefore, we can better protect fragile ecosystems, preserve biodiversity, and mitigate the adverse effects of illegal activities. At the same time, the mining industry also benefits from UAVs to increase their efficiency and revenue (Shahmoradi, Talebi, Roghanchi, & Hassanalian, 2020).

The use of drones in agriculture and environmental monitoring is continuously evolving. Ongoing advancements in drone technology, such as improved battery life, autonomous navigation, and enhanced data analysis algorithms, can further revolutionise farming practices and environmental conservation efforts. However, challenges such as regulatory constraints, privacy concerns, and technical limitations must be addressed to fully unlock the potential of drone applications in these fields.

Emergencies and disaster scenarios

In natural disasters or attacks, drones play a crucial role in an emergency response system, addressing challenges arising from limited personnel and damaged infrastructure. Their deployment enables efficient assessment of affected areas, search and rescue operations, and identifying potential threats. Many of the applications mentioned above apply to managing or avoiding disaster scenarios. Industrial facilities that often verify the integrity of their infrastructure mitigate the chance of accidents. We could even use the same system that studies wildlife in a forest to monitor a wildfire, but we must adapt these systems to the new conditions.

Rapid data collection is paramount in disaster scenarios to facilitate effective management and minimise further losses. For example, Pham, La, Feil-Seifer, and Deans (2020) uses a system with multiple drones to monitor wildfires so that we can minimise the damaged area. Similarly, Odonkor, Ball, and Chowdhury (2019) use multiple drones to monitor oil spills and map the affected area.

Besides data collection, drones can serve as delivery systems during emergencies, ensuring the transportation of vital supplies to isolated individuals. This capability is particularly significant in hard-to-reach areas where traditional means of transport may be hindered or disrupted. By leveraging drones for supply delivery, emergency response teams can quickly provide essential resources to those in need. Ackerman and Koziol (2019) present a system for blood delivery in Rwanda, which has already saved many lives.

Another critical consideration in disaster scenarios is the potential destruction of communication infrastructure. In response, drones can establish emergency communication networks (Mishra & Natalizio, 2020; Bekmezci, Sahingoz, & Temel, 2013). A drone emergency network provides temporary connectivity, enabling the deployment of other essential systems for effective disaster management. This ensures that crucial information and instructions can be transmitted to emergency response teams and affected individuals, enhancing coordination and facilitating timely assistance.

Military

Military operations use drones extensively, and we can adapt any application mentioned above to their context. The difference usually is just the fact that we invest a lot more money in the military ones. Surveillance and monitoring are essential in conflicts. Along with satellites, drones

are crucial for collecting data. Drones can identify threats such as enemies, explosives, and even dangerous chemicals on a battlefield.

Just as Rwanda has a blood delivery system, there are military systems for delivering blood and other medical supplies in combat zones (Lammers et al., 2023). The same applies to the delivery of ammunition and other supplies. But the most known delivery is explosives. We usually imagine big drones with missiles for conflicts, but small drones controlled by a simple joystick can drop grenades. They are not as destructive as missiles, but they are still very effective, cheap, and safer for soldiers.

The military also uses drones for communication. Orfanus, de Freitas, and Eliassen (2016) propose a system to establish a communication network with the help of multiple drones that self-organize according to the situation. The idea is not much different from what we envision for integrating drones with the cellular network to improve the quality of connection in our daily lives.

These military systems need to be much more robust, reliable, and secure than the ones we use daily. But the ideas are similar. The military has a lot of money to invest in research and development, and they are always looking for new technologies to improve their operations. So, we can expect to see many new applications in the future.

1.4 General challenges

All applications presented have limitations in their capabilities and deployment. Among these are the technical limitations of the drones themselves, including limited battery life, vulnerability to harsh weather conditions, payload capacity, cost, and navigation. All these challenges are addressed extensively in the literature. Many studies look to minimise energy consumption, increase battery efficiency and look for innovative recharging techniques such as harvesting energy from the environment. Drones' navigation is also a significant challenge. There are efficient methods for pathfinding, collision avoidance and many drone collaboration strategies to increase efficiency. All these are constantly improving. Drones prices are decreasing, and their capabilities are growing at an exciting rate.

The deployment of drones on a large scale requires significant infrastructure, including drone ports for takeoff and landing, charging stations, and robust communication systems for drone control and data transmission. Establishing this infrastructure, particularly in urban environments, presents a significant challenge. But many companies and countries have accepted to face this challenge.

While drones can significantly aid in environmental conservation efforts, they can also have negative impacts. Drones can disturb wildlife, especially birds, and their production and disposal can contribute to environmental pollution. We must ensure that drones cause more good than harm.

Current aviation regulations in many countries do not fully accommodate the wide range of drone applications, especially in urban environments. Drone operations have safety and security concerns in populated areas, airports' proximity, and restricted zones. Regulatory bodies must establish comprehensive guidelines allowing drone operation while ensuring safety and security. These regulations are constantly evolving but tend to be behind the technology. However, they are essential to ensure the safety of the population.

The extensive use of drones for surveillance, data collection, and delivery raises significant privacy concerns. Drones can inadvertently or purposefully capture private moments or sensitive information. Addressing these privacy issues requires a balance between the utility of drone technologies and the protection of individual privacy rights. Drones can be exploited for malicious activities, such as unauthorised surveillance, physical attacks, or cyber-attacks on the communication networks they use. Thus, implementing robust security measures is a considerable challenge. Usually, it is not random hackers that we are worried about, but the companies and the governments themselves that can use drones to spy and abuse us. This raises again the importance of regulations.

Addressing these challenges requires coordinated efforts from regulatory bodies, technology developers, and society at large. The potential benefits of drone technologies are undeniable, and their deployment on a large scale is inevitable. However, it is essential to ensure we develop and utilise these technologies responsibly to maximise their benefits and minimise their adverse effects.

1.5 Flying ad-hoc networks

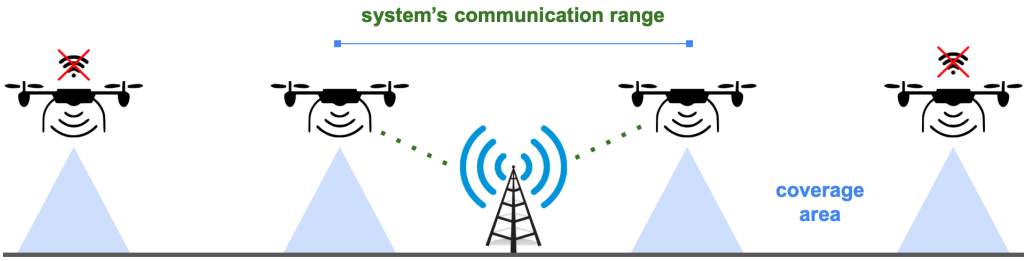
A flying ad-hoc network (FANET) is a network of drones (Bekmezci et al., 2013). A FANET is a specific type of vehicular ad-hoc network (VANET). FANETs can deploy several previously mentioned applications concerning data collection, surveillance and tracking. Some systems have several drones that rely on an existing communication infrastructure to exchange messages and are not considered a FANET. What characterises systems as FANETs is the leveraging of drone-to-drone communication.

Figure 1.2(a) shows a multi-drone system with autonomous drones and a physical infrastructure acting as a base station. Autonomous drones can leave the base station's communication range, allowing the system to cover a more extensive area. However, in this example, any information the drones need to transmit is only sent to the base station when they are within the communication range. Consequently, the drone may take a long time to send its information. Moreover, any communication between the drones must go through the infrastructure. This is not a FANET, but this approach is valid for applications that tolerate delays and non-continuous communication.

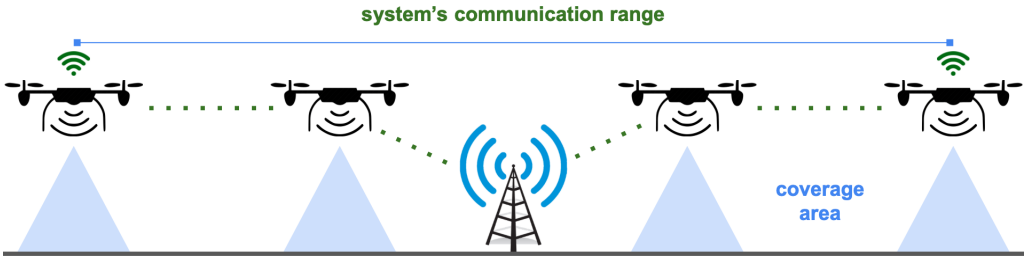
Yet, by allowing drones to communicate between themselves, we can increase the system's communication range. In Figure 1.2(b), the drones that are out of the communication range of the base station can send their messages to the nearer drones, which can relay the messages to the base station. Thus, this is a FANET.

Maintaining a communication link with the base station is a critical feature required by some applications that require continuous communication or low latency. An example is video streaming from the drones to the base station. If we consider a tracking application of mobile targets in a large area, it is essential to notify the targets' positions as soon as possible. It might be too late to know their positions only when the drones approach the base station.

FANETs have advantages such as cost. Buying several small multi-rotor drones is usually cheaper than buying one big fixed-wing drone. In some applications, by distributing the workload among the drones, it is also possible to be more time efficient. Other advantages are flexibility and adaptability. Sometimes, the system is decentralised, meaning the drones are entirely autonomous and move independently. In other cases, they depend on central orders to know where to go.



(a) Multi-drone system with no drone-to-drone communication.



(b) Flying ad-hoc network.

Figure 1.2: Advantages of FANETs.

Decentralised approaches are usually more robust to failures, but both can adapt to the environment and the application's needs over time.

In a remote location with no communication infrastructure, deploying a FANET as a temporary network is very interesting. We can do it quickly and at a low cost. During an emergency in an urban environment, such as an earthquake or an attack, communication infrastructures are expected to be damaged or overloaded. In this scenario, we can use drones to establish a temporary network to connect devices to a working base station to help manage the disaster.

1.6 Contributions

In this thesis, we look at how to efficiently place and organise drones to achieve their goals while satisfying the constraints of their applications. How we deploy drones directly impacts the efficiency of a system, and efficiency is defined differently for each application. For some, the system must maximise the surveilled area, while others must maximise the number of covered devices. Some systems must minimise the drones' energy consumption, while others must minimise the time to complete a task. We aim at defining and finding the optimal solution for the applications studied.

1.6.1 Optimal flying network for mobile sensor tracking

We consider the general case of an application that connects ground mobile sensors to a base station. The mobile ground sensors are deployed with predefined trajectories to explore an area to gather essential information. For example, consider an emergency scenario where mobile sensors help us locate victims or identify threats such as hazardous chemicals or unstable infrastructure. These sensors follow a predefined search protocol, so their trajectories are known. We aim to

establish an optimal flying network that keeps these sensors connected to a base station, and we define the network's optimality by the drones' total travel distance and energy consumption.

The main requirement of the system is continuous communication. As soon as the sensor gathers information about a victim or a threat, this information must be transmitted to the base station quickly. Therefore we want to determine the drones' position over time such that these mobile ground sensors are always covered and connected to the base station. Figure 1.3 shows this system's architecture.

We look at an optimal model to obtain the trajectories of the drones that ensure the sensors' coverage and connectivity (Dias Da Silva & Caillouet, 2020). This model is based on mixed-integer linear programming and determines the drones' positioning in a discrete space over a discrete time. By solving this linear program, we can obtain the optimal trajectories that satisfy coverage and connectivity constraints and minimise the drones' total travelled distance and energy consumption. In past optimisation models, minimising the drones' total travelled distance was equivalent to minimising their energy consumption. But considering a more precise energy model shows a trade-off such that moving more can lead to less energy consumption depending on the drone's speed.

Solving a linear program can require much time and memory, so we proposed a method based on linear programming with column generation, with which we find good-quality solutions faster than the previous method. We also leveraged the better scalability of our model to consider more positions and, consequently, find more precise answers (Dias Da Silva, Caillouet, & Coudert, 2021).

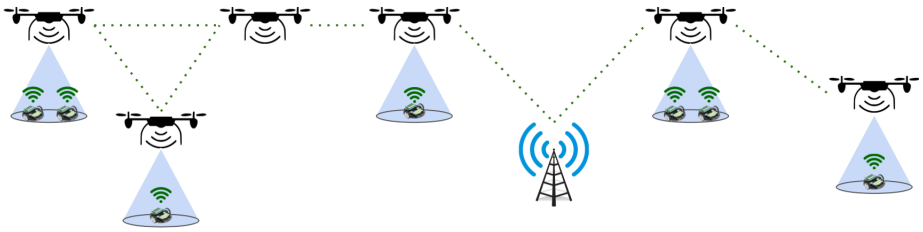


Figure 1.3: A FANET covering mobile ground sensors.

1.6.2 Charging ground sensors with drones

Sensor networks are a key component of the Internet of Things (IoT) and take part in or interact with many of the applications discussed in Section 1.3. Usually, systems that implement sensor networks are designed to work over long periods. Therefore, optimising the lifetime of each sensor, and consequently, the system itself, is a crucial element in deploying these wireless systems (Rajendran & Nagarajan, 2022). However, as much as we optimise the power consumption, these sensors eventually need to be recharged.

Recharging sensors in small systems can be done manually. However, the increasing scale of these systems (both in terms of the size of the monitored area and the growing number of sensors) means that it is no longer possible for human operators to carry out these recharges. Therefore, many energy harvesting solutions have been proposed recently, such as adding a local energy harvester system to each piece of equipment (solar panel, micro wind turbine, etc.) that allows us to harvest energy from the environment to power devices. However, these solutions are often

bulky and costly to implement. Furthermore, they are not constant in production, dependent on climatic conditions, and are not applicable in closed areas (such as large warehouses or assembly lines in the case of Industrial IoT – IIoT – for example).

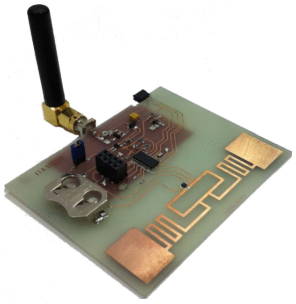


Figure 1.4: RAMSES prototype for RF power harvesting.

We focus on a wireless Radio Frequency (RF) power harvesting solution. Figure 1.4 shows an example of a module capable of harvesting radio frequencies that was proposed by De Donno, Catarinucci, and Tarricone (2014). The main advantage of RF power harvesting is we have control over when and how much energy we harvest. Moreover, we can use drones to automate the process. Drones can visit the sensors regularly to collect their data and to recharge by transmitting RF signals. In this thesis, we studied how to define flight plans for a fleet of drones to power up sensor nodes. Therefore, we propose a system with multiple drones whose goal is to charge a set of sensors. We consider that the power transmitted by drones reaches an area with a given radius and consequently can charge more than one sensor simultaneously as long as the sensors are within the transmission range. The system's goal is to minimise the time required to recharge all sensors knowing each sensor's position and energy requirement.

We expanded a state-of-the-art linear program proposed by Caillouet, Razafindralambo, and Zorbas (2019). This mixed-integer linear program can tell us which positions the drones should visit and for how long such that, in the end, all the sensors receive their required power. However, this model does not consider the efficiency loss when multiple drones simultaneously charge the same sensor. If two drones recharge the same sensor simultaneously instead of one drone, the sensor does not harvest twice the power (Altinel & Karabulut Kurt, 2016). This loss is not taken into consideration in Caillouet, Razafindralambo, and Zorbas (2019). Hence the model does not guarantee each sensor receives its required power.

We proposed a two-step solution to solve the same problem (Dias Da Silva, Busnel, & Caillouet, 2022). For the first step, we derived a mixed-integer linear program to determine where each drone should go and how long they should stay in these positions so that every sensor receives its required power. We included the ability to charge multiple sensors simultaneously with the same drone. Solving this model is the first step and gives us a list of positions each drone must visit and the required time to charge the sensors in range, but we still do not know the visitation order. Then, for the second step, we propose greedy algorithms to schedule flight plans while avoiding conflicts. This means not allowing drones to visit the same position at the same time and not allowing multiple drones to charge the same sensor simultaneously to guarantee maximum harvesting efficiency. In both steps, we aim to minimise the time to recharge the sensors, and at the end, we have a flight plan for each drone.

1.7 Publications

During this thesis, we published the following papers:

- **International conferences:**

  Dias Da Silva et al. (2022) - Trajectory Optimization for Fast Sensor Energy Replenishment Using UAVs as RF Sources - GLOBECOM - IEEE Global Communications Conference

  Dias Da Silva et al. (2021) - Optimizing FANET Deployment for Mobile Sensor Tracking in Disaster Management Scenario - International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)

- **National conferences:**

  Dias da Silva, Busnel, and Caillouet (2023) - Optimisation de Plan de Vol de Drones Autonomes Pour Une Recharge Rapide de Capteurs - Algotel 2023

  Dias da Silva and Caillouet (2022) - Quand l'immobilisme coûte plus cher que l'échange - Algotel 2022

- **Collaboration:**

  Marques, Cunha, Harada, Silva, and Silva (2021) - A Cost-Effective Trilateration-Based Radio Localization Algorithm Using Machine Learning and Sequential Least-Square Programming Optimization - Computer Communications Journal

1.8 Thesis outline

The rest of this document is organised as follows. Chapter 2 presents the background information on linear programming applied in our works. We discuss the basic definitions, complexity and duality theory. Then we present examples of the maximum flow and minimum cut problems. We also explain column generation and how to apply it to the previous examples.

Chapter 3 presents the related works to optimal deployment for FANETs. We present the energy consumption model we use in our experiments. We also look at optimal drone placement models and heuristics. Chapter 4 presents our works on deploying optimal FANETs (Dias Da Silva & Caillouet, 2020; Dias Da Silva et al., 2021).

Chapter 5 presents the related works on optimal drone deployment with energy harvesting systems. We show the energy model we use for energy harvesting. We discuss models for optimal charger placement in sensor networks which are the ground and static versions of our problem. Then we look at models that optimally place drones in networks that use energy harvesting. In addition, we look at models for optimal drone trajectories for energy harvesting. Chapter 6 presents our work minimising the time to charge sensors with drones (Dias Da Silva et al., 2022) and the improvements we have made to the model. And Chapter 7 concludes this thesis and presents future work.

Notations

API	Application programming interface
CE	Conformité Européenne (CE)
CG	Column generation
CGLP	Column generation with a linear program pricing problem
CGSP	Column generation with a shortest path pricing problem
DB	Drone-based
DB-LP	Drone-based linear program
FANET	Flying Ad-hoc Network
FCC	Federal Communications Commission
ILP	Integer linear program
LP	Linear program
LTF	Longest time first
MILP	Mixed integer linear program
MP	Master problem
NP	Nondeterministic polynomial time
RF	Radio frequency
RMP	Relaxed master problem
SB	Sensor-based
SB-LP	Sensor-based linear program
STF	Shortest time first
TOF	Time of flight
TSP	Traveling salesman problem
UAV	Unmanned aerial vehicle
WT	Wait time

CHAPTER 2

Background

In this thesis, we used graphs to represent our problems and applied linear programming methods and greedy algorithms to find the solutions. These techniques allowed us to find the optimal and near-optimal solutions to the optimisation problems defined. We employed column generation to enhance our approach's performance, enabling us to solve the linear programs faster. The notions presented here can be found in S. Lasdon (1970) and Cormen, Leiserson, Rivest, and Stein (2009).

2.1 Linear Programing

Linear programming is an optimisation model where we want to find the values for variables x_j that optimise an objective function described as a linear equation. However, these values must be such that a set of linear inequalities, called constraints, are respected. The Linear Program (2.1) has n variables and m constraints. The constants c_j describe the linear equation of the objective function as weights for the variables x_j, and the constants a_{ij}, b_i represent the linear constraints of the model. In this section, we consider variables that can assume any real positive value.

$$\begin{aligned} \text{maximise} \quad & \sum_{j=1}^{n} c_j \; x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad \forall 1 \leq i \leq m \\ & x_j \geq 0, \quad \forall 1 \leq j \leq n \\ & x_j \in \mathbb{R}, \quad \forall 1 \leq j \leq n \end{aligned} \tag{2.1}$$

A linear program can also be represented in a matrix form as shown in Linear Program (2.2). The vector x is the solution we look for, while the vector c defines the objective function and the matrix A and vector b describe the constraints.

$$\begin{aligned} \text{maximise} \quad & \mathbf{c}^\intercal \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0, \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{2.2}$$

Linear programs can solve minimisation and maximisation problems. Their constraints can be inequalities or equalities. And their variables can assume any real value, including negative ones. But the standard form of a linear program is a minimisation problem, where the constraints are all written as equalities, and the variables are nonnegative such as in Linear Program (2.3):

$$\begin{aligned} \text{minimise} \quad & \mathbf{c}^\intercal \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq 0, \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{2.3}$$

We can always represent a linear program in the standard form. We can rewrite a maximisation problem as a minimisation problem by multiplying the objective function by -1. We can represent a variable that assumes negative values as the difference between two nonnegative variables. We can add slack variables s_i to the constraints to transform inequalities into equalities such that:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ becomes } \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i$$

We do not always write linear programs in the standard form because the constraints can be easier to understand in a different form, especially without the extra slack variables. However, the methods for solving linear programs consider the standard form, and the tools that solve them usually do the conversion automatically.

You may find different definitions for the standard form of a linear program. Some say Linear Program (2.2) represent the standard form. This is not a problem when we simply apply methods and do not need to keep track of slack variables. We can always write the linear program in either form, and should choose whichever is easiest. However, when we look at the proofs of theorems, we often find assumptions that only hold for the standard form defined in Linear Program (2.3). For example, we often assume the number of variables exceeds the number of constraints $n > m$, hence $\text{rank}(A) = m$. Thanks to the slack variables, this assumption is always true for the standard form defined in Linear Program (2.3).

A feasible solution to a linear program is a set of values for x_j such that every constraint is true. If a linear program admits at least one feasible solution, it is called feasible. Otherwise, it is an unfeasible linear program. And the feasible solution that maximises the objective function is the optimal solution. In some cases, we have an infinite number of optimal solutions.

However, not every feasible linear program has an optimal solution. Suppose a linear program with a maximisation objective and no constraints. In that case, there is no limit to the maximum value for the objective function (if at least one $c_j > 0$). The same can happen even if we have constraints, and these linear programs are said to be unbounded.

These notions are easy to see in a graphical representation of the solution space of a linear program. The solution space of a linear program is the set of all feasible solutions. It is sometimes called feasible space or constraint space. We cannot see the solution space when we have many variables, but we can plot it for two variables. Figure 2.1 shows the solution space of Linear Program (2.4). Every point in the blue region delimited by the constraints is a feasible solution.

$$\begin{array}{ll} \text{maximise} & x_1 + x_2 \\ \text{subject to:} & -x_1 + 2x_2 \leq 7, \\ & 0.2x_1 + x_2 \leq 4, \\ & 2x_1 + x_2 \leq 8, \\ & x_1 \geq 0, \\ & x_2 \geq 0, \\ & x_1, x_2 \in \mathbb{R} \end{array} \tag{2.4}$$

We can immediately see that this linear program is feasible because the solution space is not empty. We can also see that the solution space is bounded, so the linear program has an optimal solution. What most of us cannot immediately see without the red point is that the optimal solution is $(2.22, 3.55)$.

Figure 2.1 also brings a few intuitions about linear programs. The first one is that the solution space is a polygon. This is true because the constraints are linear inequalities. Any solution space that is not empty nor unbounded is a polygon. The second one is that the optimal solution is at one of the polygon's vertices (an extreme point). This is not a coincidence, and it follows from Theorem 2.1.1.

Theorem 2.1.1 (Extreme point)**.** *The optimal solution appears at an extreme point of the solution space. If it appears at more than one extreme point, then it appears for all points in the segment joining any two optimal extreme points.*

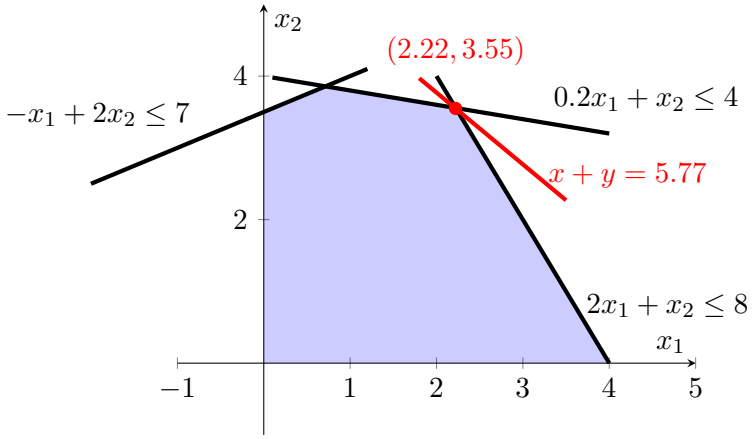


Figure 2.1: Example of solution space of Linear Program (2.4).

Proof.

First, note that the linear constraints form a polygon which is a convex set. This means if we take any two points inside the polygon, the segment that connects these points is also inside the polygon. Any point in the segment can be expressed as a linear combination of these two points. An extreme point in the convex set is a point that cannot be expressed as a linear combination of any two other points in the set.

Now, let us assume that the optimal solution x^* is not at an extreme point. This means that there are two points x_1 and x_2 in the solution space such that the optimal solution is in the segment that connects x_1 and x_2. We can express x^* as a linear combination of x_1 and x_2 as follows:

$$x^* = \lambda x_1 + (1 - \lambda)x_2, 0 \leq \lambda \leq 1$$

Since x^* is the optimal solution, we have that $c^T x^* = \alpha$ where α is the optimal value. This means we can write the optimal solution as a linear combination of the other two solutions:

$$\alpha = c^\intercal x^* = \lambda c^\intercal x_1 + (1 - \lambda)c^\intercal x_2$$

Here there are two possibilities: either $\lambda = 0.5$, in which case all these points are optimal solutions with value α, or $\lambda \neq 0.5$, in which case one of these points is a better solution than x^*, which is a contradiction.

That x_1, x_2 and x^* are optimal solutions does not imply that one is an extreme point. But in this case, we have two options again. Option 1: every point inside the polygon is an optimal solution, so the optimal solution appears at all extreme points. Option 2: x_1, x_2 and x^* are on a face of the polygon, and the extreme points of this face are also optimal solutions.

Therefore, a feasible and bounded linear program has an optimal solution at an extreme point of the solution space.

□

So linear programs can have no solution when they are unfeasible, no optimal solution when they are unbounded, and one or an infinite number of optimal solutions when they are feasible and

bounded. And the optimal solutions appear at the extreme points of the solution space. It may look simple but linear programming is a powerful tool that can solve many problems.

There are many methods to solve linear programs. The simplex method is very popular, and its main idea is to jump from one extreme point to another until it finds the optimal solution. It starts at an extreme point and moves to another extreme point, improving the objective function. It repeats this process until it reaches an optimal solution. Its worst-case complexity is exponential $O(2^n)$, where it visits all extreme points. However, it is very efficient in practice, and Spielman and Teng (2004) show that it usually takes polynomial time. Other methods can solve linear programs in polynomial time. The first one introduced by L. G. Khachiyan in 1979 is the ellipsoid method (Bland, Goldfarb, & Todd, 1981). Recently, Cohen, Lee, and Song (2020) introduced a new approach that can solve linear programs in the current matrix multiplication time. Recall that we considered only real variables in this section. When we have integer variables, things get a little more complicated.

2.2 Mixed-Integer Linear Programs

Consider the Linear Program (2.5). Notice that it is the same Linear Program (2.4) we used for the previous solution space example, but now, the variables x_1 and x_2 are integers. Therefore, we call it an integer linear program and many things change. Figure 2.2 shows its new solution space. The optimal solutions appear in red.

$$\begin{array}{ll} \text{maximise} & x_1 + x_2 \\ \text{subject to:} & -x_1 + 2x_2 \leq 7, \\ & 0.2x_1 + x_2 \leq 4, \\ & 2x_1 + x_2 \leq 8, \\ & x_1 \geq 0, \\ & x_2 \geq 0, \\ & x_1, x_2 \in \mathbb{Z} \end{array} \tag{2.5}$$

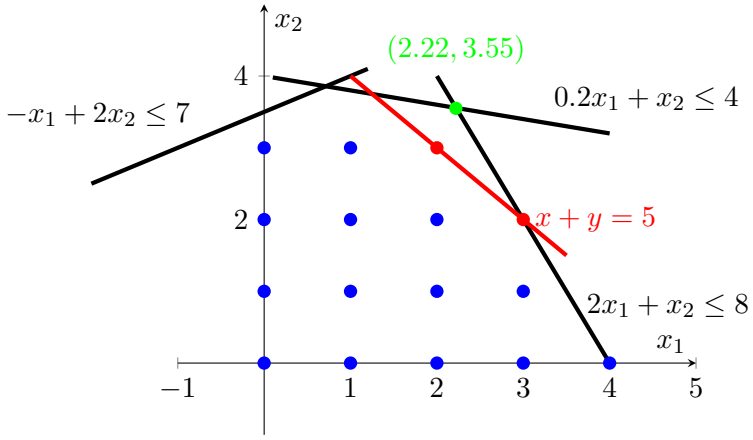


Figure 2.2: Example of solution space of the integer Linear Program (2.5).

Again, it is a feasible linear program since the solution space is not empty and is still bounded. However, the solution space is no longer convex, and finding the solution becomes more difficult.

In fact, integer linear programs are NP-hard problems. This figure also brings some intuitive ideas. First, the same maximisation problem with real variables works as an upper bound of the optimal solution. This is very useful in practice, and we refer to it as the *relaxation* of the integer linear program. Second, solving the relaxed problem and rounding the solution to the nearest integer sounds like a good plan. This can work in practice but does not guarantee an optimal solution. However, many approaches use rounding and ensure good solutions to specific problems.

Integer variables are very powerful when modelling problems. We can use them to model discrete notions, such as "yes or no", "in or out", or "how many children should we put in each bus?". For example, consider the minimum vertex cover problem, where we want to find the minimum amount of vertices that cover all edges in a graph. We can use integer variables $x_v \in [0, 1]$ for all vertices v of the graph such that $x_v = 1$ if and only if v is in the vertex cover. Real variables wouldn't model the problem well in this case because having half a vertice inside a set does not make sense.

But the variables are not always all real or all integers. We often have a mixed-integer linear program where we use both types. Solving a mixed-integer linear program is still an NP-hard problem. There are methods to find the optimal solution without verifying all possible solutions. However, it cannot always be avoided.

Branch and bound is a famous method to find the optimal solution of mixed-integer linear programs. It creates a tree of subproblems of the leading linear program. Theoretically, we must visit all tree nodes to find the optimal solution. However, this method uses upper and lower bounds to eliminate branches in this tree that are guaranteed not to wield better solutions. For some problems, branch and bound can be very efficient. As we just saw, relaxing the integer variables gives us a bound to the optimal solution. This method uses this and other heuristics to find better bounds and eliminate branches.

Morrison, Jacobson, Sauppe, and Sewell (2016) present an extensive survey on strategies used for branch and bound. The three main phases in this method are creating subproblems (branching), exploring the tree, and eliminating branches. And there are many works to improve each of them.

In conclusion, we can model many problems that would be impossible without integer variables. Moreover, finding the optimal solution for a mixed-integer linear program is difficult. But we can do it, and it is often worth it.

2.3 Duality in Linear Programming

From every linear programming problem, we can define its dual problem. The primal and dual programs are two sides of the same coin and are intimately connected through fundamental relationships. These relationships have essential implications in optimisation theory, and understanding them can help solve complex problems more efficiently. The matrix formulation shown in Linear Program (2.2) has its dual problem shown in Linear Program (2.6).

$$\begin{array}{ll} \text{minimise} & \mathbf{b}^\intercal \mathbf{y} \\ \text{subject to} & \mathbf{A}^\intercal \mathbf{y} \geq \mathbf{c}, \\ & \mathbf{y} \geq 0 \\ & \mathbf{y} \in \mathbb{R} \end{array} \tag{2.6}$$

The dual problem is the transposition of the primal problem. In the dual problem, we introduce a vector $\mathbf{y}$ with m variables, one for each constraint of the primal problem. As the matrix $\mathbf{A}$ in the primal problem has n columns (variables), the dual problem will have n constraints.

The vector $\mathbf{c}$, which previously represented the constants for the primal objective function, now serves as the vector of constants for the constraints in the dual problem. Similarly, the vector $\mathbf{b}$, which initially described the constraints of the primal problem, becomes the vector of constants for the objective function in the dual problem. Consider the following theorems:

Theorem 2.3.1 (Weak duality)**.** *If $\mathbf{x}^*$ and $\mathbf{y}^*$ are feasible primal and dual solutions, then:*

$$\mathbf{c}^\intercal \mathbf{x}^* \leq \mathbf{b}^\intercal \mathbf{y}^*$$

Proof.

This can be shown by multiplying the primal constraints by $\mathbf{y}^*$ and the dual constraints by $\mathbf{x}^*$ such that: $\mathbf{x}^{*\intercal}\mathbf{c} \leq \mathbf{x}^{*\intercal}\mathbf{A}^\intercal\mathbf{y}^* = \mathbf{y}^{*\intercal}\mathbf{A}\mathbf{x}^* \leq \mathbf{y}^{*\intercal}\mathbf{b}$

□

Theorem 2.3.2 (Strong duality)**.** *If both primal and dual have feasible solutions, then both have optimal solutions:*

$$max\ \mathbf{c}^\intercal \mathbf{x} = min\ \mathbf{b}^\intercal \mathbf{y}$$

The weak duality Theorem 2.3.1 gives us a bound on the optimal solution of the primal problem from the dual problem. The strong duality Theorem 2.3.2 states that if the primal problem has an optimal solution, then the dual problem also has an optimal solution and the optimal values of the objective functions are equal. This is very important, and in some cases, it is easier to solve the dual problem to find the optimal solution of the primal. Another interesting thing this theorem allows us to do is to verify if a solution is optimal quickly. If we have a solution $\mathbf{x}^*$ for the primal problem, we can find a corresponding dual solution $\mathbf{y}^*$. If this dual solution is feasible, it follows from Theorem 2.3.2 that both $\mathbf{x}^*$ and $\mathbf{y}^*$ are optimal solutions.

To find $\mathbf{y}^*$, we must write our linear program in canonical form. This is an extension of the standard form where m variables form a basis B. From this basis and the coefficients c_b of the objective function associated with the variables in the basis, we can find the dual solution:

$$\mathbf{y}^* = \mathbf{c_b}^\intercal \mathbf{B^{-1}} \tag{2.7}$$

Finally, we just need to verify the feasibility of $\mathbf{y}^*$ by checking if $\mathbf{A}^\intercal\mathbf{y}^* \geq \mathbf{c}$. This is how we can verify in polynomial time if a solution is optimal and how the branch and bound method knows when to stop. It also follows from these theorems that:

1. If the primal is feasible and bounded, then the dual is feasible and bounded.
2. If the primal is unbounded, then the dual is infeasible, and vice versa.
3. If the primal is infeasible, then the dual is either infeasible or unbounded.

In conclusion, understanding duality is fundamental to solving linear programs efficiently. Note that we can find a dual solution for integer and mixed-integer linear programs, and the strong duality theorem still holds to the relaxed versions of these problems. However, the solution is not necessarily integer. The difference between the integer optimal solutions x^* and y^* is called the duality gap.

2.4 Maximum flow problem

Let us look at the maximum flow problem on graphs as an example of how we can use linear programming. This problem was first introduced to model a railway traffic flow where the idea was to find the maximum amount of goods that could be transported from one city to the other. Take the example shown in Figure 2.3.

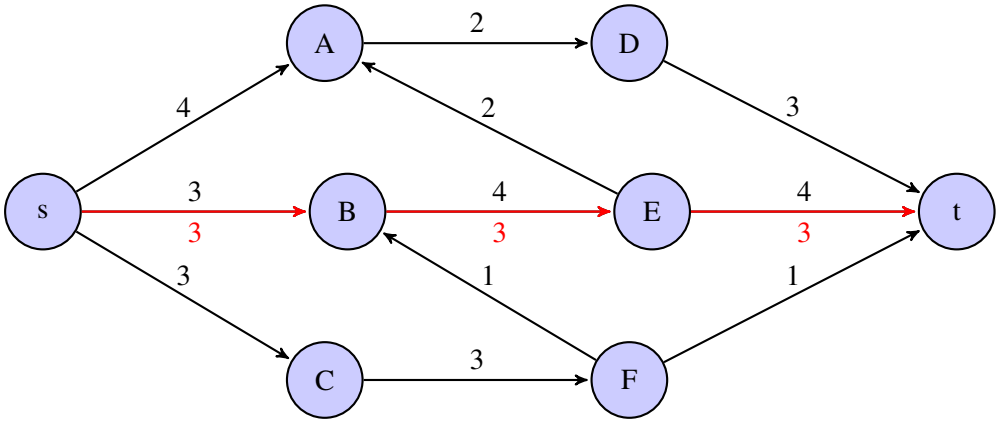


Figure 2.3: Example of a feasible flow in a graph.

We have a graph $G = (V, E)$. Suppose the vertices V are cities and the edges E are railroads between cities. Each edge $e \in E$ has a capacity c_e, the maximum amount of resources we can transport using that railroad. And our goal is to find the maximum amount of resources we can transport from city s to city t. The maximum flow problem is finding the maximum flow (resources) amount that can go from a source vertex s to a target vertex t.

There are two essential constraints in this problem. First, we cannot exceed the railroad capacity. Second, the amount of resources that arrive at a city must equal the amount that leaves the city. The exceptions are s which only produces resources, and t, which only consumes them. Figure 2.3 shows a feasible solution in red. The city s produces three resource units sent to B, then E and finally t. The railroad capacities are respected, and t receives all the resources produced by s. It is easy to see this is not the optimal solution, but a feasible one nonetheless.

To write this problem as a linear program, we define the variables f_{uv} as the amount of flow that goes from vertex u to vertex v. In Figure 2.3, the feasible solution is $f_{sB} = 3$, $f_{BE} = 3$ and $f_{Et} = 3$. All other variables are zero. The objective function of the maximum flow problem is to either maximise the amount of flow that leaves the source vertex s or the amount of flow that reaches the target vertex t:

$$\text{maximise} \sum_{v \in V \setminus \{s\}} f_{sv} \quad \text{or maximise} \sum_{v \in V \setminus \{t\}} f_{vt} \tag{2.8}$$

The first set of constraints we mentioned are the capacity constraints which state that the flow that goes through an edge must be less than or equal to the capacity of that edge. We define it for each edge as:

$$f_{uv} \leq c_{uv} \quad \forall (u, v) \in E \tag{2.9}$$

The second set of constraints say that the flow into a vertex must equal the flow out of it. The exceptions to these constraints are the source vertex that produces as much flow as needed and the target vertex that consumes as much flow as needed. We define the flow conservation constraints for each vertex as follows:

$$\sum_{v \in V} f_{uv} = \sum_{v \in V} f_{vu} \quad \forall u \in V \setminus \{s, t\} \tag{2.10}$$

The last thing we need to define in this linear program is that all flow variables can take real positive values:

$$f_{uv} \geq 0 \quad \forall (u, v) \in E, f_{uv} \in \mathbb{R} \tag{2.11}$$

Linear programming effectively models this problem, and we can find its solution in polynomial time.

2.5 Minimum cut problem

The dual problem of the maximum flow problem is the minimum cut problem. A cut is a partition of the graph's vertices into two sets S and T, such as $s \in S$ and $t \in T$. The capacity of a cut is the sum of the capacities of the edges that go from S to T. The idea is to find the minimum capacity of a cut that separates the source vertex s from the target vertex t. A cut can also be represented by a set of edges that divide the vertices into two partitions.

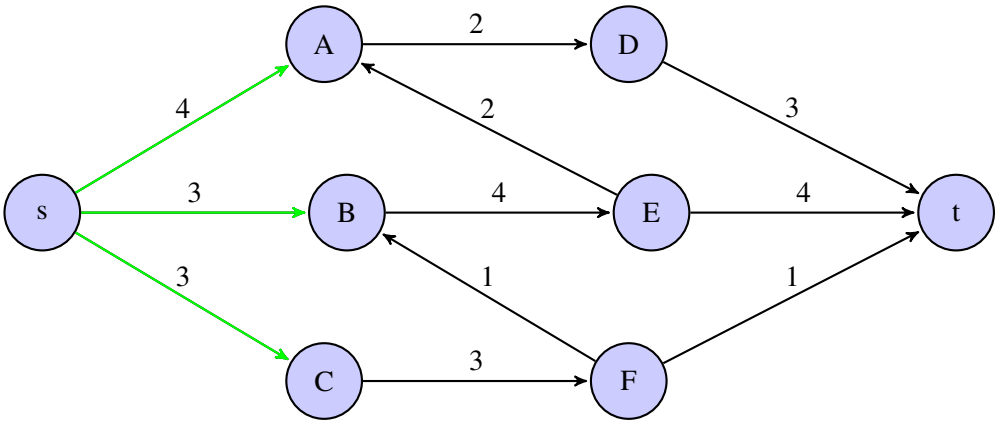


Figure 2.4: Example of a cut on a graph.

For example, in Figure 2.4, the three edges leaving the vertex s are selected, shown in green. The selection of these edges partitions the vertices of the graph into $S = \{s\}$ and $T = \{A, B, C, D, E, F, t\}$. Therefore this is a cut, and it has a capacity of $4 + 3 + 3 = 10$. Intuitively, this is an upper bound on the maximum flow value. We cannot send more than 10 units out of s. We can think of cuts as bottlenecks on the graph where the smallest bottleneck limits the maximum flow.

The following linear program models the minimum cut problem and is an example of a mixed-integer linear program. The variables y_{uv} are binary (integer $\in [0, 1]$). They represent whether the edge (u, v) is in the cut. The variables z_v are also binary and tell us whether a vertex belongs to set S. The objective function minimises the sum of the capacities of the edges in the cut.

$$\text{minimise} \sum_{(u,v)\in E} c_{uv} y_{uv} \tag{2.12}$$

$$\text{subject to: } y_{uv} - z_u + z_v \geq 0 \qquad \forall (u,v) \in E, u \neq s, v \neq t \tag{2.13}$$

$$y_{sv} + z_v \geq 1 \qquad \forall v \in V, v \neq t \tag{2.14}$$

$$y_{vt} - z_v \geq 0 \qquad \forall v \in V, v \neq s \tag{2.15}$$

$$y_{st} = 1 \qquad \text{if} (s,t) \in E \tag{2.16}$$

Whenever we have an edge $(u, v) \in E$ such that $u \in S$ and $v \in T$, or in other words, $z_u = 1$ and $z_v = 0$, this edge must belong to the cut, $y_{uv} = 1$. This is what Constraints (2.13) do. Constraints (2.14) say that for any edge that leaves the vertex s, either this vertex belongs to S or the edge is in the cut. The same idea applies to Constraints (2.15). For every edge that reaches the vertex t, either $u \in T$ or the edge is in the cut. Constraint (2.16) is an exception to the previous ones. If the edge (s, t) is in the graph, it must be in the cut.

2.6 Column Generation

Column generation is a technique used to solve linear programs with many variables (Desaulniers, Desrosiers, & Solomon, 2005). The idea is to start with just a subset of variables and then iteratively add new variables (columns) to the problem until we find the optimal solution. For a mixed-integer linear program, we apply column generation to the relaxed version of the problem. Therefore, the optimal solution found to the relaxed problem is not necessarily the optimal solution to the original problem. However, it serves as an upper bound (if maximisation) for the optimal solution to the actual problem, and in many cases, the gap between the two is small. Column generation can be coupled with the branch and bound method to find the optimal solution to the original problem.

Figure 2.5 shows the general idea of column generation. The master problem (MP) is the relaxed version of the original problem we want to solve. At the first iteration of column generation, it considers only a small subset of variables that describe the original problem and ensure a feasible solution of the MP. Once the MP is solved, we have the optimal solution regarding the small subset of variables, and our goal is to include variables to this initial subset that can improve the MP's solution.

To find these variables, we build the pricing problem from the MP dual values such that solving the pricing problem yields valid variables that can improve the MP's current solution. If the pricing problem is infeasible, the MP's solution is optimal, and we can stop the algorithm. Otherwise, we add the new variables to the MP and solve it again.

Let us use the maximum flow problem described in Section 2.4 but now in its path formulation to understand column generation better. In this formulation, instead of having flow variables for each edge $e \in E$, we have variables f_p for each possible path p in the graph from s to t. The values of these describe how much flow goes through each path. This formulation has many variables, which makes column generation a promising approach.

We define the set P as the set with all the possible paths between s and t. $P' \subseteq P$ is the subset considered in the master problem and P_e is the set of paths p such that $e \in p$ for a given edge $e \in E$. We write the master problem as:

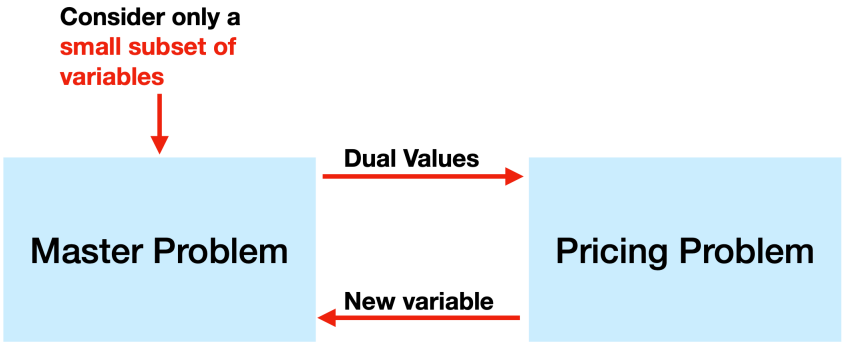


Figure 2.5: Diagram with the general idea of column generation.

$$\text{maximise} \sum_{p \in P'} f_p \tag{2.17}$$

$$\text{s.t.} \sum_{p \in P'_e} f_p \leq c_e \quad \forall e \in E \tag{2.18}$$

$$f_p \geq 0 \quad \forall p \in P' \tag{2.19}$$

The objective function is simply maximising the sum of all flows. We reformulate the capacity constraints to ensure that for every edge of the graph $e \in E$, the sum of the flow of all the paths that go through e does not exceed the edge capacity c_e as shown in Constraints (2.18). The path formulation implicitly considers the flow conservation of Constraints (2.10). For each path containing a vertex, the flow is conserved in that vertex and therefore conserved in the sum of all paths with this vertex. And just as in the previous formulation, Constraints (2.19) state that all flow variables must be positive. In the example shown in Figure 2.3, the feasible solution for the path formulation is $f_{sBEt} = 3$.

This path formulation is the master problem in our example of column generation. Instead of considering all possible paths between s and t, we start with a subset P'. This subset P' must be enough to find at least one feasible solution. This is easy in this problem since $\mathbf{0}$ is a feasible solution, and we know the answer to the problem is bounded since it can be, at most, the sum of all edges' capacities.

Once we find the optimal solution to the master problem over P', we look at the dual problem defined as:

$$\text{minimse} \sum_{e \in E} c_e y_e \tag{2.20}$$

$$\text{s.t.} \sum_{e \in E} y_e \geq 1 \quad \forall p \in P' \tag{2.21}$$

$$y_e \geq 0 \quad \forall e \in E \tag{2.22}$$

We define the dual variables y_e for each constraint in Constraints (2.18). Therefore, one variable for each edge of the graph. These variables represent whether an edge belongs to a cut or

not. The objective of the dual problem is to find the cut, or set of edges, with minimum capacity as defined in Objective Function (2.20). This is the same minimum cut problem described in Section 2.4 but in its path formulation. Constraints (2.21) state that for any path from s to t, $p \in P'$ must contain at least one edge selected. These constraints ensure that a feasible solution to the dual problem is a proper cut that divides the graph between s and t.

From the optimal solution to the master problem, we have the dual values y_e. The next step is to produce the pricing problem. The pricing problem aims to find a new variable y_p that renders the dual problem infeasible. An infeasible dual problem means that the current solution to the master problem is not optimal, and this new variable can improve it. So the goal is to find a path $p \in P \setminus P'$ that breaks its corresponding constraint in Constraints (2.21). We define the pricing objective as follows:

$$\text{minimise} \sum_{e \in E} y_e w_e \tag{2.23}$$

The dual values y_e are constants in the pricing problem, and we introduced the binary variable $w_e, \forall e \in E$. This variable is 1 if the edge e belongs to the path we are looking for. Otherwise, it is 0. We need to find a set of edges that form a valid path in the graph from s to t, and that minimises the Objective Function (2.23). If this path with minimum value is not enough to break the dual constraint in Constraints (2.21), then we have found the optimal solution. We define the following constraints to ensure that the path is valid:

$$\sum_{e=(s,v) \in E} w_e = 1 \tag{2.24}$$

$$\sum_{e=(v,t) \in E} w_e = 1 \tag{2.25}$$

$$\sum_{e \in E^+(v)} w_e - \sum_{e \in E^-(v)} w_e = 0 \quad \forall v \in V \setminus \{s,t\} \tag{2.26}$$

The first Constraint (2.24) ensures that we choose exactly one edge leaving vertex s. The second Constraint (2.25) ensures that we choose exactly one edge entering vertex t. And Constraints (2.26) say that for any other vertex, the number of selected edges entering and leaving this vertex must be the same. This integer linear program is our pricing problem. However, it is simply a shortest-path problem where the dual values are weights for the edges. Since the dual values are positive, we can solve it in polynomial time.

Hence, the column generation for this example is reduced to: solve the MP', and obtain the dual values y_e. Use the dual values as weights on G and search for the shortest path. If this path weighs less than one, add it to P' and solve the master problem again. Otherwise, if we cannot find a path that weighs less than one, the current dual solution is feasible for all P. Therefore, the master problem's current solution is optimal.

Column generation is a powerful technique that we can apply to many problems. It is interesting when the number of variables is too large. In cases where we can solve the pricing problem in polynomial time, such as this example, column generation becomes even more efficient. In cases where the pricing problem is NP-hard, we can use heuristics to speed up the process.

For mixed-integer linear programs, it does not ensure an optimal solution. However, it can find the optimal solution in many cases and good solutions in others. We can use column generation

with the branch and bound method to find the optimal solution. Moreover, with duality theory, we can verify the optimality gap to understand how far a solution is from the optimal one. This gap is the distance between our best feasible solution and the optimal solution of the relaxed problem. We are usually satisfied with a solution that has a small optimality gap since finding the optimal solution with a branch and bound method can be expensive.

2.7 Solvers

There are many solvers for linear programming. Some of them are open-source, and others are proprietary. A known open-source solver is the GNU Linear Programming Kit (GLPK) (*GLPK - GNU Project - Free Software Foundation (FSF)*, n.d.). Two proprietary solvers commonly used are Gurobi (*Gurobi Optimizer*, n.d.) and CPLEX (*IBM Documentation*, 2021). In this thesis, our experiments use CPLEX, which is free for academic use (*CPLEX Optimization Studio Is Free for Students and Academics!*, n.d.). Every model we present is implemented using the CPLEX API, which calls the CPLEX solver. The CPLEX solver uses state-of-the-art methods to obtain the optimal solution efficiently. Its code is written in C and is highly optimised. We mention the CPLEX version, API, and machine specifications used in our experiments (Sections 4.1.3, 4.2.4 and 6.4).

PART

Optimal flying network for mobile sensor tracking

CHAPTER 3

Related works to optimal FANET deployment

One of the applications we studied in this thesis is the deployment of FANETs to connect mobile sensors with the base station. We consider a scenario where we know the coordinates of mobile ground sensors. These sensors move inside a known region and collect critical data, which a base station must receive as soon as possible.

Drones can form a flying network to relay the data from the sensors to the base station. We want to model this problem to obtain the drones' optimal trajectories. The metrics we aim to minimise are the travelled distance and the energy consumption. Minimising travel distance is essential as high mobility is prejudicial to communication protocols. Moreover, minimising energy consumption is crucial to increase the system's autonomy.

This chapter looks at the related works to optimal FANET deployment. We choose our energy model to better define an optimal solution to our problem. More precise energy models show lower speeds do not always minimise energy consumption. Then, we review how other works approach drones' deployment optimisation for covering mobile sensors and for connectivity. We focus on the results that obtain optimal solutions according to a cost function and ensure the drones cover all sensors. However, optimal methods tend to focus on the network topology instead of drones' trajectories.

3.1 Energy consumption model

In this thesis, we often estimate the energy consumption of a drone. There are many models to do this, and we adopted the one presented by Zeng, Xu, and Zhang (2019) where the power consumption $\mathrm{Power}(v)$ depends on the drone's speed v and is defined in Equation (3.1). Table 3.1 shows the definition and values for the constants.

$$\mathrm{Power}(v) = \underbrace{\mathrm{Power}_0 \left(1 + \frac{3v^2}{U_{tip}^2}\right)}_{\text{BladeProfile}} + \underbrace{\mathrm{Power}_i \left(\sqrt{1 + \frac{v^4}{4v_0^4}} - \frac{v^2}{2v_0^2}\right)^{1/2}}_{\text{Induced}} + \underbrace{\frac{1}{2} d_0 \rho s A v^3}_{\text{Parasite}} \tag{3.1}$$

$$\mathrm{Power}_0 = \frac{\delta}{8} \rho s A \Omega^3 R^3$$

$$\mathrm{Power}_i = (1 + k) \frac{W^{3/2}}{\sqrt{2\rho A}}$$

The blade profile power is required to turn the rotor in the air. The induced power is the power necessary to overcome gravity. Parasite power is the one the drone consumes because of the drag of the fuselage, vibrations, air displacement and other factors. This model ignores the additional energy consumption caused by acceleration and deceleration. This model is more complex than most works consider, but it is also easily reproducible thanks to the values shared by the authors.

Figure 3.1 shows the power consumption behaviour per speed that can be counter-intuitive to most people. We expect a higher speed to induce a higher power consumption, and this is true. However, there are speeds at which the drone consumes less power than it would if it was hovering ($v = 0$). Adjusting its rotors' speed to maintain position is more demanding than moving forward slowly. However, slow is a relative notion. The minimum power consumption happens at $v^* = 10.2\ \mathrm{m/s}$, which is fast enough for many applications. Some works assume that minimising

Variable	Definition	Value
W	Aircraft weight (N)	20
ρ	Air density ($\mathrm{kg/m^3}$)	1.225
R	Rotor radius in m	0.4
A	Rotor disc area in $\mathrm{m^2}$	0.503
Ω	Blade angular velocity ($\mathrm{rad/s}$)	300
$Utip$	Tip speed of the rotor blade ($\mathrm{m/s}$)	120
s	Rotor solidity	0.05
d_0	Fuselage drag ratio	0.6
k	Incremental correction factor to the induced power	0.1
v_0	Mean rotor induced velocity when hovering	4.03
δ	Profile drag coefficient	0.012

Table 3.1: Constants used in the energy consumption model.

the drone's movements will minimise energy consumption, but this is not always true. Therefore we adopt this more realistic model to estimate the energy consumption of the drones in our experiments.

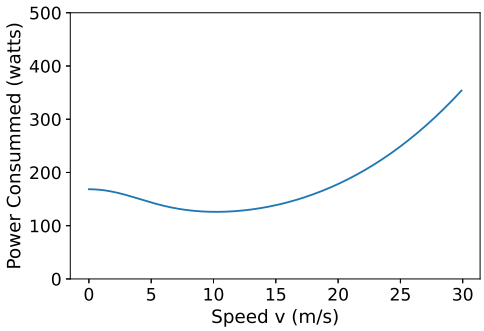


Figure 3.1: The consumption of power per speed.

In most of our experiments, the drones travel at $10.2\ \mathrm{m/s}$, the speed with minimum power consumption. The total energy consumed during the flight is $\mathrm{Power}(v)$ multiplied by the time of the flight. Whenever there is an exception, we specify how we decide the drones' speed.

3.2 Optimal coverage of mobile targets

Zorbas, Di Puglia Pugliese, Razafindralambo, and Guerriero (2016) propose a model to position drones to monitor targets optimally. They start with a static case of the problem and then extend it to a dynamic version. Their models ensure all the drones monitor all targets at every moment. Both static and dynamic models also avoid simultaneously placing drones in the same position. They minimise the energy cost and the number of drones to deploy. To do so, they define an integer linear program (static version) and a mixed-integer non-linear program (dynamic version). In addition, the authors propose efficient heuristics to achieve suitable solutions.

First, let us look at the static version of their model. It considers a discrete space for the drones' deployment P. For each drone u and each position $(x, y, h) \in P$, a binary variable δ^u_{xyz} represents if u is in (x, y, h) or not. So a feasible solution to the model describes a drone deployment in P. The targets are on the ground with arbitrary known positions. The drones monitor a circular area below them that increases with their altitude. However, the authors set a height limit ($h_{\max}$ depends on camera resolution) where the drones cannot recognise the targets if they go higher. The constraints of the static model ensure that each target is within the monitoring radius of at least one drone. The first objective function they proposed is to minimise the number of drones used:

$$\min \sum_{(x,y,h)} \sum_{u \in U} \delta^u_{xyz} \tag{3.2}$$

The second objective function aims to minimise energy consumption. To define this objective, Zorbas, Di Puglia Pugliese, et al. (2016) estimate the power a drone consumes to leave the ground and hover at a height h for t seconds as follows:

$$E = (\beta + \alpha h)t + \frac{h}{s}\text{Power}_{\max} \tag{3.3}$$

β is the power consumed when the drone hovers. α is a rotor speed multiplier and depends on the physical properties of the drone, such as weight and propeller characteristics. $\text{Power}_{\max}$ is the maximum power the motor consumes, and s is the drone's speed to reach height h. The first term is the energy consumed while the drone hovers in place, and the second is the energy consumed to reach height k. The authors assumed this model was accurate enough for their purposes based on drone manufacturers' information, simulations, and experimentations with ground mobile robots' motors. The higher the drone is, the more energy it consumes, but at the same time, the bigger its coverage radius. So by consuming more energy, drones can cover more targets, reducing the number of deployed drones.

Therefore, they apply Equation (3.3) to Equation (3.2) and obtain the second objective function as follows:

$$\min \beta \sum_{(x,y,h)} \sum_{u \in U} \delta^u_{xyz} t + \alpha \sum_{(x,y,h)} \sum_{u \in U} h\delta^u_{xyh} t + \frac{\text{Power}_{\max}}{s} \sum_{(x,y,h)} \sum_{u \in U} h\delta^u_{xyh} \tag{3.4}$$

With this, they model the static version of their problem as an integer linear program. Their model for the dynamic version uses discrete time with steps Δ. Each target moves during a period $[t_s, t_e]$, and their positions at each time step are known. They represent each original moving target as set a set of static targets that appear for a single time step where the original should be at that time. They add real variables τ^u_{start} and τ^u_{end} to represent when a drone u starts working and stops. Their constraints ensure at each time step, all targets are inside the monitoring radius of an active drone. The dynamic model has many more variables and constraints as it must consider all time steps. Also, in Equation (3.4), the time is no longer a constant t, but a difference of variables $\tau^u_{\text{end}} - \tau^u_{\text{start}}$. Therefore the problem is no longer linear, and finding its solution is computationally expensive. Although a smaller model, the static version is still computationally expensive as it is an integer linear program.

Given the proposed optimisation models' complexity, the authors propose heuristics to solve both versions. The heuristic for the static problem first places one drone on top of each target

and then merges drones to improve the objective function. Whenever two drones are close, the algorithm transforms them into a single drone that monitors all targets of the original two drones. The new drone's position and altitude are determined accordingly. The merge is validated if the objective function improves and the new height does not exceed the limit $h_{\max}$. Then, the algorithm erases redundancies to eliminate unnecessary drones. The greedy heuristic for the dynamic version uses the static heuristic for the first time step. Then at each time step, adapts the drone altitude according to the target's movements and reevaluates if a merge or split is necessary.

These methods are computationally efficient and achieve good results. The authors also propose other interesting heuristics, including a decentralised one. We can adapt the notion of static and dynamic targets to many applications. For example, we can consider communication coverage for mobile devices instead of a camera with a monitoring radius depending on the resolution.

Their model can be simplified. They have binary variables δ^u_{xyh} for every drone and position. We can replace them with binary variables δ_{xyh}. After all, each drone visits only one position, and each position is visited by at most one drone. The first is a constraint, and the second is a consequence of how their variables are defined. Each drone's start and finish time variables can be associated with the positions instead. So we can achieve the same results with fewer variables. Nonetheless, the heuristics presented are not affected by this, and the models' complexity is still high.

However, their models do not optimise the drone's trajectories. As the targets move, the number of drones required to cover them varies, and the positions needed to monitor them also change. Their models optimise the drones' deployment as these changes happen but not the drones' movement. One of the constraints they define is that we can only deploy a drone at most in one position:

$$\sum_{(x,y,h)} \delta^u_{xyh} \leq 1, \forall u \in U \tag{3.5}$$

Therefore, their model evolves in time with drones that "appear" and "disappear". It does not reuse a drone after it stops working at τ^u_{end}. A new drone is deployed if a new position needs a drone after τ^u_{end}. For example, suppose we have one single target. The target spends 30 seconds in a position and then moves to a new position, where it stays for the rest of the observation period. Their model will deploy one drone at each location when only one drone is necessary to monitor this target. Their heuristics do consider the inactive drones when reassigning drones to new positions. But the model does not optimise the drones' movements here.

Moreover, to their energy model, both the deployment of these two drones or a single drone that follows the target are equivalent. This is false, and finding the optimal drone trajectories is a more complex problem. In this thesis, we want to model how drones can change their positions optimally (minimum energy consumption) while still achieving the application's goals and satisfying its requirements.

3.3 Optimal coverage of mobile targets with limited drone batteries

In the same year, Pugliese, Guerriero, Zorbas, and Razafindralambo (2016) extended their models proposed in Zorbas, Di Puglia Pugliese, et al. (2016). The new model considers the drone's battery and replaces the drones when their batteries are empty. They model these replacements by adding non-linear constraints in the mathematical formulation. They also present heuristic procedures based on mixed-integer programming.

This model considers the same energy consumption shown in Equation (3.3). The authors introduce the energy limit of each drone as a non-linear variable to the previous model that counts how many drones are needed to hover a position during $\tau^u_{\text{end}} - \tau^u_{\text{start}}$. If this period is 1.5 times the drone's battery life, we need two drones to cover that position.

They transform their non-linear model into a mixed-integer linear program by adding auxiliary linear variables. It is still computationally expensive, but they propose a heuristic that divides this problem into more easily solved subproblems. The heuristic solves these subproblems at each iteration (time step). Still, as in the previous work, drone trajectories are not optimised.

Optimal network for mobile sensors

Caillouet, Giroire, and Razafindralambo (2019) used a similar approach as the previous models to solve a more restricted problem. They considered a set of mobile ground sensors, and the drones' mission is to connect every sensor to the base station at each time step. Not only must every sensor be in communication range with (covered by) at least one drone, but also, the drones must form a communication network between every sensor and a base station. In the previous models, we can consider targets as sensors and replace the monitoring range for a communication range between sensors and drones. This approach would tell us how to place drones to collect sensor data optimally. However, Caillouet, Giroire, and Razafindralambo (2019) consider a scenario where a base station must receive the sensor data immediately, so the drones must form a communication path between the sensors and the base station.

First, they proposed a mixed-integer linear program that finds the optimal positions to deploy the drones at each time step. The linear program minimises the number of deployed drones' global distance to the base station, reducing their total consumed energy. Then, they used column generation to solve the problem more efficiently.

Their mixed-integer linear program is similar to the previous work (Zorbas, Di Puglia Pugliese, et al., 2016), but it is a simpler model while still adding the notion of connectivity to the problem. The authors define the 3D space as a discrete set of positions P and model the problem as a graph $G^t = (V^t, E^t)$. Because the ground sensors are mobile, the graph changes for each time step. Hence, the index t indicates how the graph looks at each time step t. The period of observation is $[0, T]$ where T is the number of time steps.

The graph's vertices are the base station, the positions P, and the ground sensors. An edge between two vertices means they are in communication range. So an edge between two positions means that if drones are deployed in these positions, these drones can communicate with each other. Moreover, an edge between a position and a sensor indicates that a drone deployed in that position can cover (communicate with) the sensor. Interestingly, selecting positions such that all sensors are covered and form a communication path with the base station is equivalent to finding a flow from the base station to each sensor in this graph.

The goal is to minimise the number of drones used and their energy consumed. Hence the authors define the binary variables z^t_p. This variable is one if a drone is in p at time t and zero otherwise. They associate these variables with the flow such that, at time t, a flow can only pass through a position p if a drone is deployed on this position at this time.

Solving their mixed-integer linear program means selecting positions to deploy drones to cover all sensors and form a communication path with the base station. However, this model does not consider the drones' trajectories. It does not determine which drone is in which position. Therefore

we cannot know the drones' movements, only the network's topology. The same problem we face with the previous models.

The authors consider that the power a drone consumes is proportional to the distance it travels. Hence, their objective function is to minimise the number of drones deployed and the difference in selected positions from one time step to the other as follows:

$$\min \left(\sum_{t=0}^{T} \sum_{p \in P} C_p z_p^t + \max_{t \in [1,T]} \left| \sum_{p \in P} C_p z_p^t - \sum_{q \in P} C_q z_q^t \right| \right) \tag{3.6}$$

The first term represents the number of positions used, and the second is the maximum difference between subsequent time steps. Minimising this maximum difference does reduce the drones' movements but is not equivalent to minimising them. Even if it were, we saw in Section 3.1 that minimising the drones' movements is not equal to minimising the drones' energy consumption.

Given the complexity of solving this model, the authors proposed a column generation approach. As in the example shown earlier in Section 2.6, the authors find a pricing problem they can solve as a shortest path problem, allowing column generation to obtain near-optimal solutions much faster.

This model considers all the constraints we are interested in and can be solved efficiently. However, it fails to represent the drones' trajectories. It does reduce drones' movements but does not minimise them, nor it minimises energy consumption.

3.4 Heuristic for sensors coverage and connectivity

Mahoro Ntwari, Gutierrez-Reina, Toral Marín, and Tawfik (2021) approach a very similar problem. They want to deploy drones to cover all (fixed) ground sensors. Instead of a path from each sensor to the base station, they ensure a connection between any two sensors. They also ensure at least p drones cover each sensor as redundancy benefits the network. They divide their approach into three steps: clustering to elect possible deployment positions, an integer linear program to choose the deployment positions, and a greedy algorithm to connect the sensors to the base station.

Instead of dividing the 3D space as a grid of deployment positions as Caillouet, Giroire, and Razafindralambo (2019) did, they argue that we should determine this set according to the sensors' positions. We can find better solutions by tailoring this set's shape to the input. Therefore, the grid partitioning approach is not suitable for this problem. Instead, they propose an initial step with a clustering approach.

They use a single-pass clustering algorithm. The algorithm takes the first sensor on the list, creates a cluster, and then adds all sensors in range (radius of drone's coverage). It updates the cluster representative whenever it adds a sensor. The cluster representative is a deployment position for a drone to cover all sensors within the cluster. This position is the clusters' centroid. At the end of the algorithm, cluster representatives are the deployment positions to ensure coverage of all sensors.

The authors mention a disadvantage of using this algorithm: the deployment positions depend on the order of the sensors on the list. Therefore, this ordering impacts the final solution, which is not ideal. However, it is still better than the grid partitioning approach. One alternative suggested by the authors is to use the number of clusters determined with this algorithm as a parameter for

a more complex clustering algorithm, such as k-Means, which can lead to better clusters and does not depend on the list ordering.

They also argue about the disadvantages of hard clustering in this scenario (when each sensor belongs to only one cluster). Having overlapping clusters to have more choices in the next phase is important. Few options can lead to worse results. Therefore they consider a smaller coverage range to find more clusters (deployment positions). However, they must balance having enough choices to find a better solution in the next step but not too many choices that solving the optimisation problem becomes too hard. Hence, they apply the elbow method to estimate the number of clusters needed in the next phase. Nonetheless, this clustering approach proves better than simply dividing the space into a grid as it elects more meaningful deployment positions related to the sensor's positions.

Once the clustering algorithm generates the possible deployment positions, they solve the integer linear program for the set covering problem. Solving this problem means selecting the minimum number of deployment positions to cover all sensors. However, this does not ensure a communication path from each sensor to the base station.

Therefore, the authors propose a greedy algorithm to provide the connectivity. They look for the shortest distance between two disconnected components of the graph and add a vertice between them. This vertice might be enough to connect them, or it might be a new disconnected component. The algorithm repeats until the graph is connected.

This does not provide the optimal solution. It reduces the number of drones deployed but does not minimise it. Moreover, it does not consider the distance drones travel or energy consumption. Neither can it be used to obtain the drones' trajectories in a mobile scenario. We could repeat their approach for each time step, but as the solution proposed by Caillouet, Giroire, and Razafindralambo (2019), we would only know the positions the drones' must visit and not which drone should be in which position. Nonetheless, it is efficient, gives reasonable solutions, and is much more scalable than the previous models.

CHAPTER 4

Column generation for optimal FANETs' trajectories

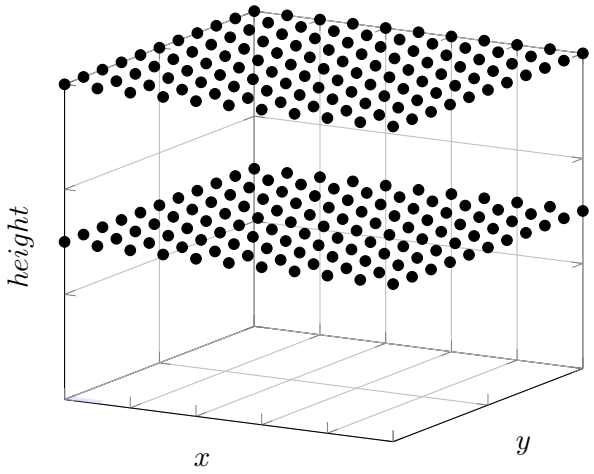


Figure 4.1: Example of a grid partitioning approach to determine the set P.

In this Chapter, we deploy a FANET to collect data from mobile sensors on the ground. The mobile sensors follow a predefined search strategy, so their moves are known. We must ensure a communication path between each sensor and the base station.

In Section 4.1, we present our first work (Dias Da Silva & Caillouet, 2020). We extend the model of Caillouet, Giroire, and Razafindralambo (2019) and propose a mixed-integer linear program to find the optimal FANET deployment. We minimise the drones' total travelled distance and energy consumption and determine a fair optimal solution to balance both metrics. However, the complexity of the model makes it impractical to solve it for large scenarios.

Hence, in Section 4.2 we present our second work (Dias Da Silva et al., 2021). We propose a column generation approach to solve the same problem. We show that column generation can find good solutions much faster than the previous approach. Moreover, we can leverage our scalability to find more precise deployments.

4.1 Optimal trajectories for FANET deployment

Consider we have T time steps. Inside a squared region, we know the coordinates of all mobile sensors at each time step. The set S^t contains these coordinates for time step t and is an input to this model. We aim to deploy drones to connect all sensors to the base station and minimise their trajectory cost. In our experiments, we represent the base station as b, which is the origin of the coordinate system. However, the model can consider the base station in any position, a mobile base station, and multiple base stations.

We define the set P as the possible deployment positions in the 3D space over the square region where the sensors move. We obtain P by dividing the 3D space as a grid. Figure 4.1 shows an example, the sensors move on the plane xy, and we have the deployment positions above. The notation P_b refers to the possible deployment positions plus the base station $P_b = P \cup \{b\}$. We consider the drones to have a communication range R_c and a coverage radius r_p. So two drones in positions p and q can communicate if they are within the communication range R_c:

$$\text{distance}(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2 + (h_p - h_q)^2} \leq R_c \tag{4.1}$$

While R_c is a fixed constant in our model, r_p depends on the height of the drone's position $p \in P$ and the drone's directional antenna half beam-width (visibility angle) θ:

$$r_p = h_p \tan \frac{\theta}{2} \tag{4.2}$$

Therefore, the coverage radius increases with the drone's altitude. A drone deployed in position p covers sensor s if the distance between the sensor and the drone projection on the ground is less than r_p:

$$\text{distance}(p_{xy}, s) = \sqrt{(x_p - x_s)^2 + (y_p - y_s)^2} \leq r_p \tag{4.3}$$

We define a graph $G^t = (V^t, E^t)$ for each time step t. The vertices are the base station, the possible deployment position, and the sensors. The edges represent whether two vertices are within the communication range:

$$\begin{aligned} V^t =& \{b\} \cup P \cup S^t \\ E^t =& \{(p,q), \forall p,q \in P_b | \text{distance}(p,q) \leq R_c\} \cup \\ & \{(p,s), \forall p \in P, s \in S^t | \text{distance}(p,s) \leq r_p\} \end{aligned} \tag{4.4}$$

The graph is undirected. An edge exists between two positions in P or between a position and the base station if they are within the communication range R_c. For future works that want to be more specific, we can use a different communication range between the base station and the positions. An edge between a position and a sensor exists if the sensor is inside the coverage radius r_p.

4.1.1 Mixed-integer linear program

As in the previous model (Caillouet, Giroire, & Razafindralambo, 2019), we use flow variables f^t_{pq} to represent the flow between two vertices at time step t. We ensure that at each time t, a flow equal to the number of sensors ($|S^t|$) leaves the base station b. This flow is conserved in each position $p \in P$. And each sensor consumes one unit of flow. Hence, it represents a communication path between the base station and each sensor.

$$\begin{gathered} \sum_{(p,q) \in E^t} f^t_{pq} - \sum_{(q,p) \in E^t} f^t_{qp} = \begin{cases} |S^t| & \text{if } p = b \\ 0 & \text{if } p \in P \\ -1 & \text{if } p \in S^t \end{cases} \\ \forall t \in [0,T], p \in V^t,\ f^t_{pq} \in \mathbb{R} \end{gathered} \tag{4.5}$$

We define the binary variable z^t_p to represent whether or not a drone is deployed in position p at time t. We tie this variable to the flow variables such that whenever a flow passes through a position p, it forces the deployment of a drone in it:

$$f^t_{pq} \leq z^t_p |S^t|, \quad \forall t \in [0,T], p \in P, q \in V^t | (p,q) \in E^t \tag{4.6}$$

However, this is not enough to model the drones' trajectories. We must know where each drone is in each time step. For this, we have the set U with the available drones we can use. Then,

we define the binary variables z^t_{pu} to represent whether or not a drone u is in position $p \in P_b$ at time t. Hence, we also consider the drones positioned at the base station. However, these variables complicate things, and we need to add constraints to ensure our solutions are valid.

First, we need to ensure at most one drone is in a position at each time step to avoid collisions:

$$\begin{aligned} z^t_p &= \sum_{u \in U} z^t_{pu} \\ \forall t \in [0, T], p \in P_b & \\ z^t_p \in [0, |U|] \text{ if } p = b & \\ z^t_p \in [0, 1] \text{ if } p \neq b & \end{aligned} \tag{4.7}$$

Note that Constraints (4.7) avoid more than one drone at the same place at each time step but do not avoid possible collisions between time steps when the drones move. We do not consider this in our model.

Second, we need to ensure that the drones do not disappear or appear in multiple positions at the same time. Constraints (4.8) ensure a drone is either at the base station or at one position in P at each time step:

$$\sum_{p \in P_b} z^t_{pu} = 1 \quad \forall t \in [0, T], u \in U \tag{4.8}$$

Figure 4.2 shows an example of how the flow variables force the deployment of drones. On a time step t, the graph has 5 vertices: the base station b, deployment positions p and q, and sensors s_1 and s_2. The edges are shown in green. The flow must pass through positions p and q to reach the sensors. Hence, drones u_1 and u_2 are deployed.

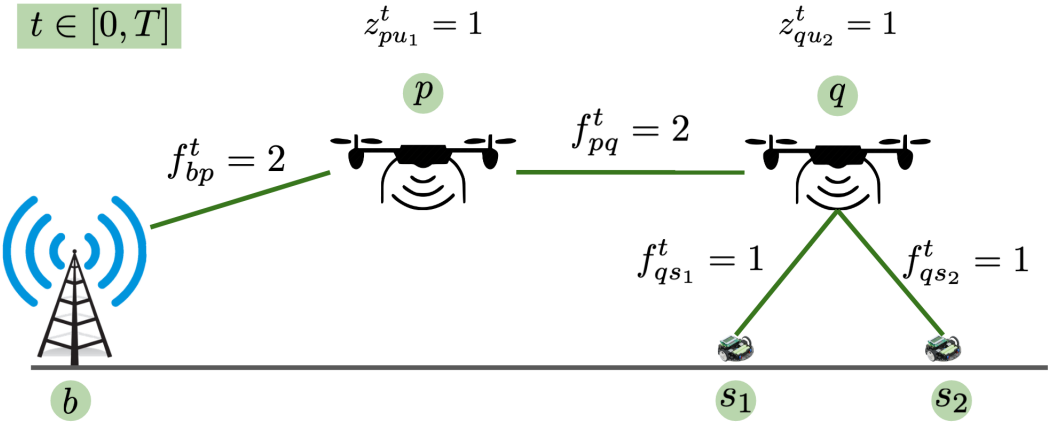


Figure 4.2: Example of how the flow variables force the deployment of drones.

Now, we can know where each drone is at each time. However, if we want to minimise the distance the drones travel or the energy consumption, we need to represent the notion that "a drone u was in p and moved to q from time $t-1$ to t". We can do this with the multiplication of two binary variables z^{t-1}_{pu} and z^t_{qu}. If this multiplication is one, then the drone u was in p at time $t-1$ and moved to q at time t. Otherwise, that is false. But we need to represent this with linear

constraints. Therefore, we introduce the binary variable z^t_{pqu} to represent $z^t_{pqu} = z^{t-1}_{pu} z^t_{qu}$ with the following constraints:

$$z^t_{pqu} \leq z^{t-1}_{pu} \tag{4.9}$$
$$z^t_{pqu} \leq z^t_{qu} \tag{4.10}$$
$$z^t_{pqu} \geq z^{t-1}_{pu} + z^t_{qu} - 1 \tag{4.11}$$
$$z^{t-1}_{pu}, z^t_{qu}, z^t_{pqu} \in [0,1]$$
$$\forall t \in [1,T], p, q \in P_b, u \in U$$

4.1.2 Objective functions

With the definition of z^t_{pqu}, we can account for the cost C_{pq} of going from p to q. Hence, we can define $f(z, C)$ as the function that returns the cost of a trajectory z with the cost function C:

$$f(z,C) = \sum_{t=1}^{T} \sum_{p,q \in P_b} \sum_{u \in U} C_{pq} z^t_{pqu} + \sum_{p,q \in P_b} \sum_{u \in U} \left(C_{bp} z^0_{pu} + C_{pb} z^T_{pu} \right) \tag{4.12}$$

In $f(z, C)$, the first term considers for every time step $t \in [1, T]$, the cost for the drones coming from their previous positions to the current one. That's why we exclude $t = 0$ from the sum. The second term considers the cost of the drones going from the base station to their first position and from their last position to the base station.

The first cost we consider is the distance ($C_{pq} = \text{distance}(p, q)$) travelled by the drones, which the following objective function minimises:

$$\text{minimise } f(z, \text{distance}) \tag{4.13}$$

The second cost we consider is the energy consumption of the drones. We adopt the model of Zeng et al. (2019) presented in Section 3.1 to estimate the energy consumption. They model the energy consumption of a rotary-wing drone as a function of its speed $\text{Power}(v)$ shown in Equation (3.1). Using their model, we divide the drones' energy consumption into two terms: E_{mov} that is the energy to reach position q, and E_{hov}, that is the energy to hover in q after it arrives:

$$E_{pq} = E_{\text{mov}} + E_{\text{hov}} \tag{4.14}$$
$$E_{\text{mov}} = \text{Power}(v) \frac{\text{distance}(p,q)}{v}$$
$$E_{\text{hov}} = \text{Power}(0) \left(\Delta_t - \frac{\text{distance}(p,q)}{v} \right)$$

The drones have a time Δ_t to move from p to q between each time step. It means that if a drone needs to go from p to q between time steps, it has to travel at least at a speed of $v_{\min} = \text{distance}(p, q)/\Delta_t$ to be able to do it on time. If it travels faster than $v_{\min}$, it has to hover for the remaining time of Δ_t once it arrives. Moreover, Zeng et al. (2019) determine the speed $v^* = 10.2$ m/s as the speed with minimum power consumption. However, this speed is not always optimal in our model because we consider E_{hov}. So we define the speed v as:

$$v = \max\left(v_{\min}, v_{\text{opt}}\right) \tag{4.15}$$

Whenever $v_{\min}$ is lower than v^*, we do a numerical search for $v_{\text{opt}} \in [v_{\min}, v^*]$ that minimises E_{pq}. However, if $v_{\min}$ is higher than v^*, then we have to use $v_{\min}$ to ensure the drones' positioning at each time step.

There are a few exceptions we need to take care of. First, if the drone remains in its position, we consider $v = v_{\min}$ to avoid the 0 division, and we obtain simply the hovering energy during Δ_t. Second, whenever the drone comes from or goes to the base station, we consider $E_{\text{hov}} = 0$. This is because we assume drones land at the base station. And when they take off, they can time their departure to arrive at q at the end of Δ_t. With this, we use the cost $C_{pq} = E_{pq}$ to define the second objective function:

$$\text{minimise } f(z, \text{energy}) \tag{4.16}$$

Instead of considering only the energy consumption or the total travelled distance, we introduce a third objective function that considers both. To do so, we introduce a parameter α, which is the weight of each objective function:

$$\text{minimise } (1-\alpha) f(z, \text{distance}) + \alpha\beta f(z, \text{energy}) \tag{4.17}$$

The parameter β is a normalisation factor that ensures the energy consumption and the distance travelled have the same order of magnitude. We define it as $\beta = v^*/\text{Power}(v^*)$, where $\text{Power}(v^*)$ is the drones' minimum power consumption.

4.1.3 Results

With this model, we can find the drones' trajectories that minimise the distance travelled, the energy consumption, or a combination of both. And we ensure that all sensors can communicate with the base station at each time step. We implemented it using the Java API of IBM Cplex solver 12.10.0 in a computer with an Intel(R) Core(TM) i7-7700HQ CPU 2.80 GHz, 8 Gb RAM, running Ubuntu 18.04.4 LTS operating system. The memory required to find the optimal solutions can be very high, so we set an 80Gb swap file.

As for the default parameter values in our experiments, we have: the number of deployment positions varies between 9, 16 and 25 positions distributed as a grid in a $100\text{m} \times 100\text{m}$ square at a height of 45 m. We use $U = 5$ drones with a 60 m communication range to cover 5 mobile sensors. The observation period was 7 time steps, with the default time between time steps set as $\Delta_t = 2$ seconds. The sensors move according to the random waypoint model, and we vary their speeds between 5 and 20 m/s.

Even these small scenarios can require much time and memory to find the optimal solution. The number of deployment positions changes the dynamic graph's size and impacts the number of variables and constraints of the mixed integer linear program. The same happens for the number of time steps. The positions' height also has a big impact. Higher heights give us a bigger coverage radius for the drones. If the drones have a bigger coverage radius, we need fewer drones to solve the problem, and they can move less. We also chose a big communication range to reduce the number of drones necessary to build the path from the base station to the sensors.

The objective here is to present optimal solutions for small scenarios, allowing us to highlight drone behaviour when we choose to optimise its total flying time or its total energy consumed.

Results with one objective

We computed the trajectories that minimise the total travelled distance ($\alpha = 0$) for the 9 traces of the mobile sensors using 9, 16 and 25 deployment positions each, totalling 27 experiments. The average total travelled distance per experiment for 9 deployment positions was 642 m while the average with 25 deployment positions was 568 m. This happens because the more positions available, the more precise the drones' movement. We can verify it by looking at the total distance travelled during the observation period, which is the total distance minus the distance for the deployment and return phases, shown as the partial distance in Table 4.1.

$\|P\|$	α	Distance (m)	Partial distance (m)	Energy (J)	Time
9	0	642.62	41.78	36389.42	3.5 s
16	0	645.24	21.85	47050.98	5.46 min
25	0	568.20	20.03	48514.66	2.82 h
9	1	956.03	443.19	16558.69	12.6 min

Table 4.1: Average time to solve the mixed-integer linear program for different numbers of positions and α.

Although increasing the number of positions leads to more precise movements, it also impacts the resolution time. Table 4.1 shows the average resolution time to find the optimal solution that increases exponentially with the number of positions.

When minimising the energy consumption ($\alpha = 1$), we only used 9 deployment positions and all the default parameter values. As shown in Table 4.1, the average distance travelled during the observation period goes to 443.19 m, which is ten times higher than when we minimise the total travelled distance. However, the average energy consumed is reduced to 16558.69 Joules, at least two times lower than in the distance optimisation.

Using the power consumption model of Zeng et al. (2019), it is clear that minimising the drones' altitude or total travelled distance is not the same as minimising their energy consumption, and our model takes this into account. For example, in Figure 4.3, we have the deployment of 4 drones in two subsequent time steps while drone 1 remains at the base station (BS[1]). The sensors marked as green "x"s are all covered by the drones, whose coverage range is shown in red, and they move in $100\text{m} \times 100\text{m}$ square region. Moreover, we see the communication path formed by the blue edges. The sensors move between the time steps, but the network's topology remains valid, so the drones do not need to move to satisfy our constraints. Nonetheless, drones 3 and 5 and drones 2 and 4 exchange positions between time steps to minimise energy consumption.

Trade-off analysis

Here we analyse the trade-off between minimising the total travelled distance or the total energy consumed. We fix the number of deployment positions as 9. All other parameters assume their default values. We found the optimal solution for 9 different traces of the mobile sensors varying the value of α from 0 to 1 for each one of them. Figures 4.4(a) and 4.4(b) show the trade-off between the two objectives.

When minimising both the travelled distance and the energy consumption, we find intermediaries solutions that are good with values not far from the optimal ones for both metrics. For example, when $\alpha = 0.5$, the solutions have a total travelled distance of 683.34 m and consumed

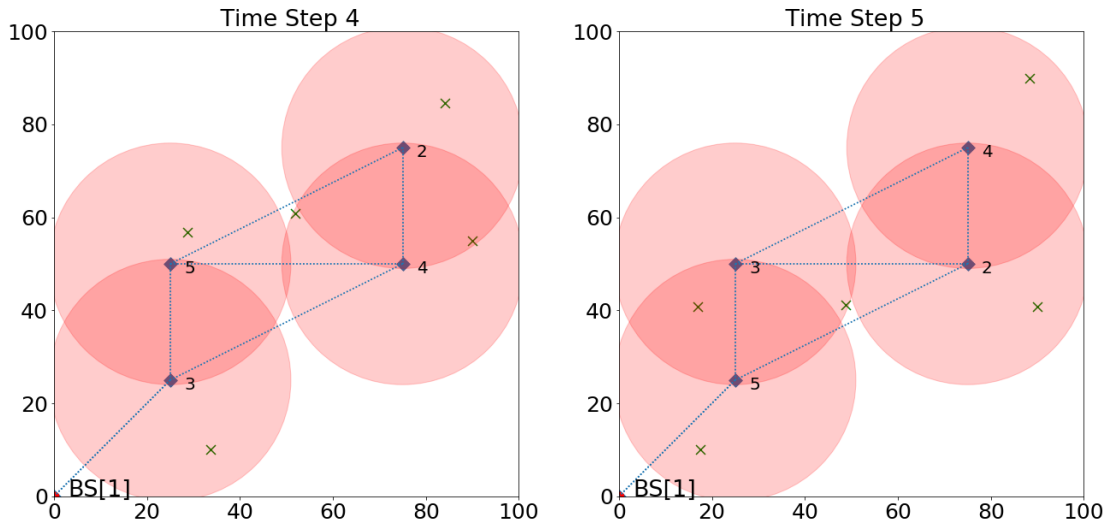
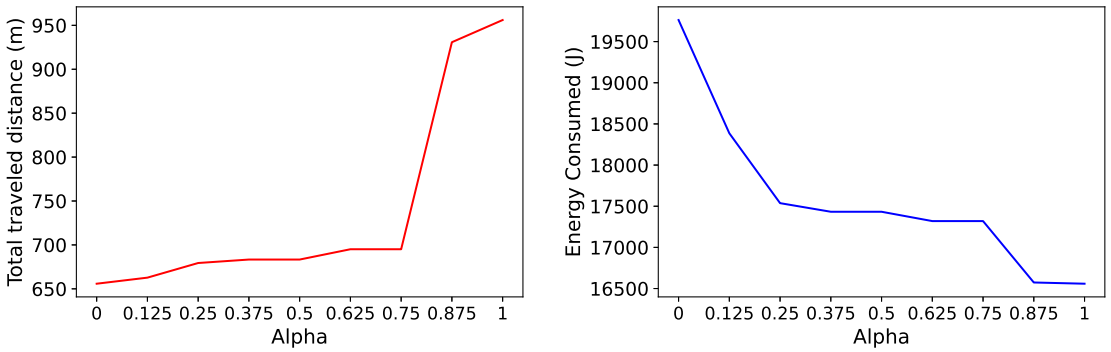


Figure 4.3: Drones changing position between time steps to save energy.

energy of 17432.84 J on average. This is a good compromise that fairly balances distance and energy.



(a) Total travelled distance.

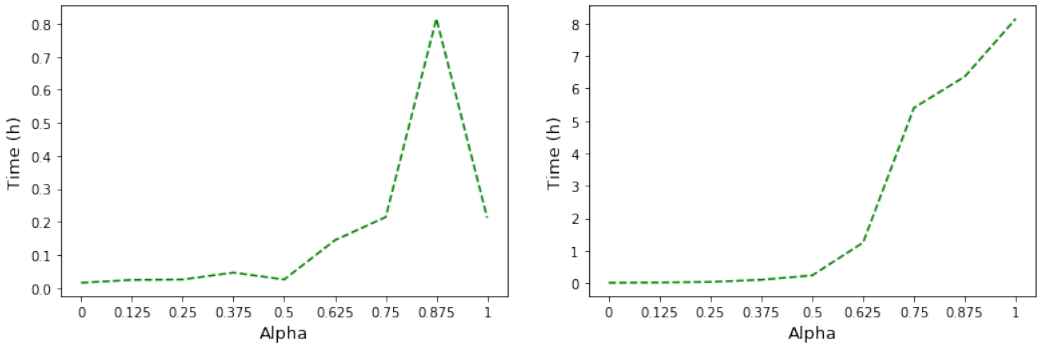
(b) Total energy consumption.

Figure 4.4: Trade-off between total travelled distance and energy consumption.

Resolution time

Mission accomplished, except for one detail. The complexity of solving this mixed-integer linear program is very high. In Table 4.1, we have the time to solve the model as we increase the number of positions. It quickly becomes impractical to solve it.

We observed that CPLEX needs more time to find the optimal solutions for energy consumption than it needs to find the optimal solutions for the total travelled distance. We believe this is a consequence of the higher mobility that makes it harder for CPLEX to execute the branch and bound method. We can see this in figure 4.5(a), where we vary the value of α. When we minimise only the total travelled distance with $\alpha = 0$ the average solution time is 54 seconds, but when



(a) Solution time for $\Delta_t = 2$.

(b) Solution time for $\Delta_t = 3$.

Figure 4.5: Mobility impact on solution time.

we minimise only the energy consumption with $\alpha = 1$, the average goes to 12.6 minutes. Moreover, increasing the time between time steps to $\Delta_t = 3$ increases the drones' mobility as they can travel further with the optimal speed. This increases the solution time even more, as we see in Figure 4.5(b).

4.2 Applying column generation

Column generation often allows us to solve problems faster, and Caillouet, Giroire, and Razafindralambo (2019) successfully applied it to their model. Therefore, we investigated whether it could help us with our extended model. As we discussed in Chapter 2, we need to define a master problem and a pricing problem to use column generation. The master problem is the linear program that chooses the optimal trajectories for each drone but considers only a limited version of the problem. The pricing problem is responsible for identifying which variables the master problem should include in its consideration to find better solutions.

We rewrite the previous model of Section 4.1 in a path formulation, meaning each possible trajectory a drone can follow is a binary variable. Since there is an exponential number of possible trajectories in our search space, it is inefficient to enumerate all of them and solve the program with all the possible variables. Hence, we start by considering only a small number of variables allowing a feasible solution for our problem, and we use the pricing problem to add more variables (trajectories) to the master problem that can lead us to better solutions.

4.2.1 Master problem (MP)

We use the same graph defined in Equation (4.4). The set C is the set of all possible paths (trajectories) a drone can follow during the observation period T. A path is a sequence of positions $p \in P_b$ through which the drone passes each time step. The set P is a grid as in the example of Figure 4.1.

We assume that the drones travel between positions in a straight line, and we do not consider collisions in the time Δ_t between time steps. The variables that describe the master problem are:

- y_c is a binary variable for each path $c \in C$ that is 1 if c is selected by a drone and is 0 otherwise.

- f_{pq}^t is a real variable between two positions that are within communication range of each other (i.e. satisfying Equation (4.1)). It represents the flow at time t from a drone at position p to a drone at position q. Once again, the flow determines in which positions the drones are deployed to form the network.

Despite the exponential number of variables, we take advantage of column generation and start by considering just a small subset C' of possible paths. Then, we use the pricing problem to add new paths to C'. Each trajectory $c \in C'$ has constants δ_{cp}^t that describe it. δ_{cp}^t is 1 if trajectory c goes through the position p at time t and 0 otherwise. Our master problem is described with Objective Function (4.18) and Constraints (4.19) to (4.23).

$$\text{minimise} \sum_{c \in C'} D_c y_c \tag{4.18}$$

$$-\sum_{c \in C'} \delta_{cp}^t y_c \geq -1 \qquad \forall t \in [0,T], p \in P \tag{4.19}$$

$$\sum_{(p,q) \in E^t} f_{pq}^t - \sum_{(p,q) \in E^t} f_{qp}^t = |S^t| \qquad \forall t \in [0,T], p = b \tag{4.20}$$

$$\sum_{(p,q) \in E^t} f_{pq}^t - \sum_{(p,q) \in E^t} f_{qp}^t = 0 \qquad \forall t \in [0,T], p \in P \tag{4.21}$$

$$\sum_{(p,q) \in E^t} f_{pq}^t - \sum_{(p,q) \in E^t} f_{qp}^t = -1 \qquad \forall t \in [0,T], p \in S^t \tag{4.22}$$

$$|S^t| \sum_{c \in C} \delta_{cp}^t y_c - \sum_{(p,q) \in E^t} f_{pq}^t \geq 0 \qquad \forall t \in [0,T], p \in P \tag{4.23}$$

We aim to choose a set of trajectories for the drones that build a valid FANET during the period T. If the drones follow the selected paths, the sensors will be covered and connected to the base station at each time step. The objective function in Objective Function (4.18) is to minimise the total cost of the selected paths. Here we consider the total distance of path c as the cost D_c of each trajectory:

$$D_c = \sum_{(p,q) \in c} \text{distance}(p,q)$$

Constraints (4.19) ensure that we cannot choose more than one path going through the same position p at the same time t. The base station is not included in this equation since it can simultaneously hold as many drones as possible. Although Constraints (4.19) avoid multiple drones in the same position at the same time, they do not include collision avoidance between the time steps. Constraints (4.20) to (4.22) are the same as Constraints (4.5) in the previous model. They ensure the base station communicates with every sensor.

Constraints (4.23) are the path formulation equivalent to Constraints (4.6). They ensure that if there is flow going through a position p at time t, then there must be a path c selected that goes through p at time t. In other words, we must select a path c that places a drone at p at time t. Otherwise, no flow is allowed to go through p.

Solving this path formulation mixed-integer linear program is equivalent to solving the formulation presented in Section 4.1. We obtain the trajectories with minimum cost that connect every

sensor to the base station at each time step. One advantage of this formulation is that we do not need to consider the set U of available drones as before.

This is our master problem. We consider relaxed variables as we saw in Chapter 2. Hence, y_c can take any real value between 0 and 1. The initial set C' of paths is $\{y_p, \forall P_b\}$ such that y_p is a trajectory that hovers position p in all time steps. If we select all these paths, we will have a feasible solution.

4.2.2 Pricing problem

To derive our pricing problem, we look at the dual problem. A variable of the dual program corresponds to a constraint of the primal problem and vice versa. We define the dual variables $\beta_{tp}^{(4.19)}$ and $\beta_{tp}^{(4.23)}$ for Constraints (4.19) and Constraints (4.23) respectively. The dual constraint for variable y_c is as follows.

$$-\sum_{t=0}^{T}\sum_{p\in P}\delta_p^t\beta_{tp}^{(4.19)} + |S^t|\sum_{t=0}^{T}\sum_{p\in P}\delta_p^t\beta_{tp}^{(4.23)} \leq D \tag{4.24}$$

Once we solve the master problem, we obtain the corresponding values for all $\beta_{tp}^{(4.19)}$ and $\beta_{tp}^{(4.23)}$ and Inequality (4.24) is satisfied for all paths in C'. This follows from the strong duality (Theorem 2.3.2).

Then, we need to find path $c \in C \setminus C'$, which violates Inequality (4.24). If such a path exists, we add it to the current set C' of variables of the master problem. If no new trajectory violates Inequality (4.24), it follows from the strong duality theorem that we have found the optimal solution to the relaxed master problem. We propose two pricing problems: (i) a mixed-integer linear program (CGLP) and (ii) a shortest path algorithm (CGSP).

4.2.2.1 Mixed-integer linear program pricing (CGLP)

The pricing problem variables must describe a path that violates Inequality (4.24). We use the following variables:

- δ_p^t is a binary variable that is 1 if the new path goes through p at time t and is 0 otherwise.
- D is a real variable representing the new path's total distance.
- ω_{pq}^t is a binary variable that is 1 if the new path contains the position p at time $t-1$ and the position q at time t and it is 0 otherwise.

First, we need to ensure a feasible solution to the pricing problem is a valid path. Consequently, the Equation (4.25) ensures we only select one position in each time step:

$$\sum_{p\in P\cup\{b\}}\delta_p^t = 1, \quad \forall t \in [0,T] \tag{4.25}$$

We can interpret the variables ω_{pq}^t as the multiplication of δ_p^{t-1} and δ_q^t. Hence, just as we did for z_{pqu}^t, we represent this multiplication with the following Equations (4.26) to (4.28):

$$\omega_{pq}^{t} \leq \delta_{p}^{t-1} \tag{4.26}$$
$$\omega_{pq}^{t} \leq \delta_{q}^{t} \tag{4.27}$$
$$\omega_{pq}^{t} \geq \delta_{p}^{t-1} + \delta_{q}^{t} - 1 \tag{4.28}$$
$$\forall t \in [1, T], p, q \in P_b$$

Equation (4.29) ensures that the variable D corresponds to the new path total distance to verify the objective function in Equation (4.30):

$$\sum_{t=0}^{T} \sum_{p,q \in P_b} D_{pq} \omega_{pq}^{t} + \sum_{p \in P_b} (\delta_{p}^{0} D_{pb} + \delta_{p}^{T} D_{bp}) = D \tag{4.29}$$

We reformulate the dual constraint in Inequality (4.24) as the objective function of the pricing problem:

$$\text{maximise} - \sum_{t=0}^{T} \sum_{p \in P} \delta_{p}^{t} \beta_{tp}^{(4.19)} + |S^{t}| \sum_{t=0}^{T} \sum_{p \in P} \delta_{p}^{t} \beta_{tp}^{(4.23)} - D \tag{4.30}$$

The objective of Equation (4.30) is to break the optimality of constraint of Inequality (4.24). Suppose we can find a solution with a positive objective value for our pricing problem. In that case, we break the optimality constraint and find a new variable to add to our master problem, allowing us to find a better solution.

By satisfying these equations, we have a valid path. If we can find a path with a positive value for the objective function in Equation (4.30), we add it to C' and solve the relaxed master problem again. If we cannot break the optimality constraint, we cannot find a better solution to our relaxed master problem.

Column generation can help us reach the solution faster because solving the relaxed master problem can be done in polynomial time. The pricing problem is an integer linear program, and we need to solve it at each iteration of column generation. But it is smaller than the mixed-integer linear program proposed in Section 4.1 and can be solved faster. Moreover, after the last iteration of column generation, we need to solve the mixed-integer master problem. However, the subset C' tends to be small so that we can reach our final solution fast.

Nonetheless, as the number of positions and time steps increases, this pricing problem becomes bigger and slows down the column generation. We can improve this by substituting the pricing problem as an integer linear program with an algorithm that reaches the same results in polynomial time.

4.2.2.2 Shortest-path algorithm for pricing (CGSP)

We can look at our pricing problem as a shortest-path problem. Take a graph $G = (V, E)$ as shown in Figure 4.6. The vertices are the positions and the base station at each time step, and the edges connect past positions to future positions. We can interpret our pricing problem as finding a path from b_i to b_f with the shortest cost. This means selecting a sequence of positions through time, allowing us to break the optimality constraint. For each edge, we consider the cost as the negative distance between the positions connected by the edge and for each vertex, we consider the cost as

$\beta_{tp}^{(4.19)} - |S^t|\beta_{tp}^{(4.23)}$. Note that in Equation (4.30), the goal is to maximise the cost, so we invert the sign of the cost here to have the equivalent shortest path problem. Finding the path with the shortest total cost in this graph is equivalent to solving the previous pricing problem.

Lemma 4.2.1 (Complexity of CGSP)**.** *The CGSP pricing problem can be solved in time* $O(|P|^2|T|)$.

Proof.

Let us build the graph $G = (V, E)$ as shown in Figure 4.6 with one vertex per position (including the base station) and per time step. So vertex (p, t) represents the position p at time step t. Then, there is an arc from vertex (p, t) to vertex $(p', t+1)$ representing position p' at time step $t+1$ if position p' is reachable from position p. Furthermore, there is a vertex b_i in G with arcs to each vertex $(p, t = 0)$ for $p \in P$, and a vertex b_f with arcs from each vertex $(p, t = T)$ for $p \in P$ to d. Vertex b_i (resp. b_f) models the fact that the initial (resp. final) position of a drone is on the base station.

The cost of an arc is the negative distance between the positions it connects, and we set the cost of a vertex (p, t) to $\beta_{tp}^{(4.19)} - |S^t|\beta_{tp}^{(4.23)}$. The costs of b_i and b_f are null.

Clearly, the graph G is acyclic, and so computing the shortest path from b_i to b_f can be done in time $O(|V| + |E|)$ (Cormen et al., 2009). Since the number of vertices is in $O(|P||T|)$ and the number of arcs is in $O(|P|^2|T|)$, the time complexity follows.

Finally, observe that the shortest path from b_i to b_f in G is a minimum cost trajectory.

□

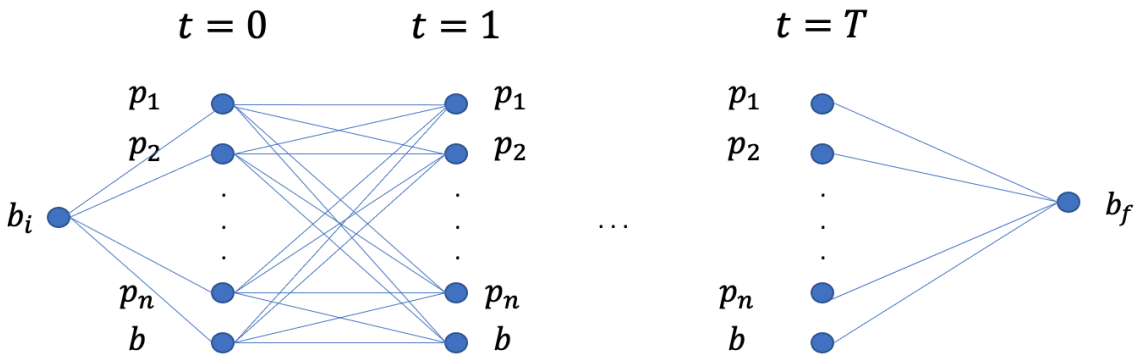


Figure 4.6: The pricing problem as a shortest path problem.

4.2.3 Solving the column generation

To solve the column generation (CG), we start with a subset C' of possible paths to obtain a feasible solution for the master program. Then, we solve the relaxed master problem (RMP), the master problem in which we relax the binary variables y_c to accept real values in range $[0..1]$. After, we solve the pricing problem using $\beta_{tp}^{(4.19)}$ and $\beta_{tp}^{(4.23)}$, the dual variables of the master.

If we obtain a positive value for the objective function of the pricing problem, the solution describes a new path that we must add to C'. After adding this new path, we repeat the process by solving the RMP with the new variable.

Otherwise, if we obtain a negative value for the objective function, it means there are no more paths that we need to add to C'. We then solve the integer master problem (MP) with C', where the binary variables y_c can only be 0 or 1, and we get our final solution.

Recall that the set C' is not necessarily enough to find the optimal solution to the original problem in C. However, it does happen that it contains the optimal solution. And in any case, this is a valid solution whose cost $\tilde{z}_{\mathrm{MP}}$ is an upper bound of the optimal solution. Furthermore, the objective value z^*_{RMP} obtained when solving the RMP is a lower bound on the optimal solution value. Hence, we can evaluate the so-called optimality gap of a solution as $(\tilde{z}_{\mathrm{MP}} - z^*_{\mathrm{RMP}})/z^*_{\mathrm{RMP}}$. We will see in our experiments that this gap is small in practice, meaning the solutions are of good quality. We would need to combine the CG with a branch-and-bound method to obtain an optimal solution consistently.

4.2.4 Results

The models presented were implemented using the Java API of IBM Cplex solver 12.10.0 and run on a computer equipped with an Intel(R) Core(TM) i9-10900K CPU 3.70 GHz and 64 GB RAM, running Fedora 33 operating system. The inputs for our models are the mobile ground sensors' trajectories inside an area A over a period T. In our experiments, we consider A to be a 10000 m^2 square with five sensors, and the observation period consists of 7 time steps with 2 seconds between them. The sensors' coordinates are generated as a random walk, where at each time step, the sensors move at a fixed speed of 5 $\mathrm{m/s}$ in a random direction inside the closed area. Given the moves of the sensors, we solve our column generation model to find the trajectories of the drones. We analyse the model's efficiency with both pricing approaches, the linear program and the shortest path algorithm. We make the input parameters vary, and for each considered scenario, we average the results obtained in ten instances.

We distribute the possible positions as a grid dividing the area A into squares of equal length. Each intersection in this grid corresponds to a possible deployment position in the set P. In our experiments, all possible positions are 45 m above the ground by default.

4.2.5 Scalability

We first show that our CG model is scalable compared to a compact linear model presented in Section 4.1. In Figure 4.7, we compare the time required to solve the mixed-integer linear program (MILP) proposed from Section 4.1, our column generation model with the standard pricing problem as a linear program (CGLP) and the pricing problem as a shortest path problem (CGSP). Solutions for more than 25 positions were impossible to obtain for the MILP, but we can reach 64 positions for CGLP with a long resolution time. Both column generation models are faster than the MILP, while the shortest path pricing problem provides a much faster alternative. Considering 64 positions, the shortest path pricing problem reduces the average solution time from 15.54 hours to 13.35 minutes. This validates our scalable approach for FANET planning in rescue situations. As expected, increasing the number of positions quickly increases the problems' complexity.

4.2.6 Optimality

However, as explained in Section 4.2.3, the column generation does not always reach the optimal solution. Nonetheless, the column generation reaches the same solution as the MILP in 54% of our experiments. The average optimality gap of column generation was 5%. When considering 9

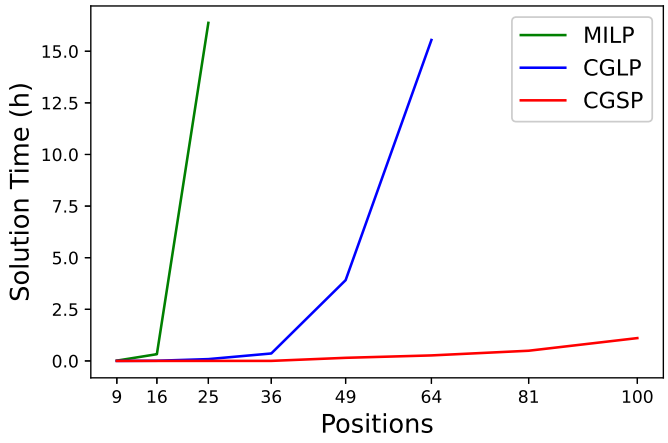


Figure 4.7: Comparison between the time to solve the problem when considering the compact mixed-integer formulation (MILP) and the column generation using either the linear program pricing problem (CGLP) or the shortest path pricing problem (CGSP).

positions, the average optimality gap was 9%, but as we increased the number of positions to 16 and 25, the average optimality gap decreased to 2% and 3.1%, respectively. This happens because it is necessary to combine column generation with a branch-and-bound process to compute the optimal integer solution.

Although column generation does not always reach the optimal solution, the distance to the optimal solution is small, and we can consider more positions than with the MILP. Moreover, increasing the number of positions improves the solutions thanks to the more precise positioning of the drones. With few positions, whenever a sensor leaves the coverage of a drone, the drone needs to travel a bigger distance than necessary to keep covering the sensor since the positions are far apart from each other. Once we increase the number of positions, the drones can be deployed more precisely such that they need to move less, and if they need to move, they can adjust their positioning by moving smaller distances due to the higher density of positions.

This is shown in Figure 4.8, where the more positions, the less the drones travel on average. Therefore, if we look at the column generation with the shortest path pricing problem (CGSP), when considering 64 positions, we find better solutions than the MILP considering 25 positions. Not only are the answers of better quality, but we also find them faster.

4.2.7 Pricing policy

We tested several policies to improve our CG resolution when solving the shortest path pricing problem. Given that we may have many paths per iteration with minimal cost that can be added to C' (shortest path is not unique in the graph of Figure 4.6), we explore adding multiple paths at the same time to C' between two resolutions of the RMP to add several columns in the same iteration and try to accelerate the resolution. Figure 4.9(a) shows that increasing the maximum number of paths that can be added per iteration seems to be interesting when we consider a few positions, as the average time required to solve the problem is shorter. However, as the number of positions increases, adding more paths per iteration leads to a longer time to solve the problem. This happens because the more positions we consider, the more solutions can be found for the

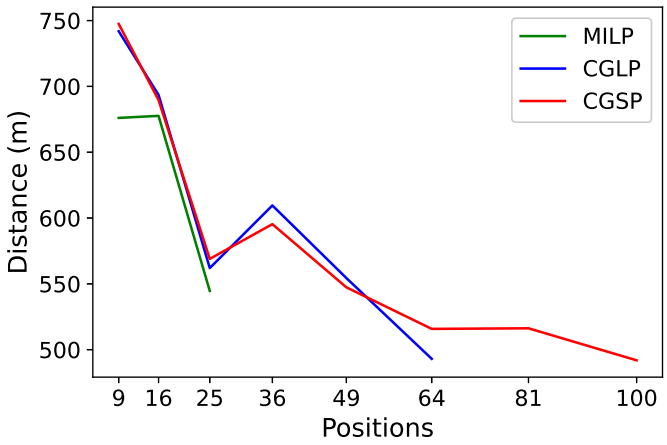
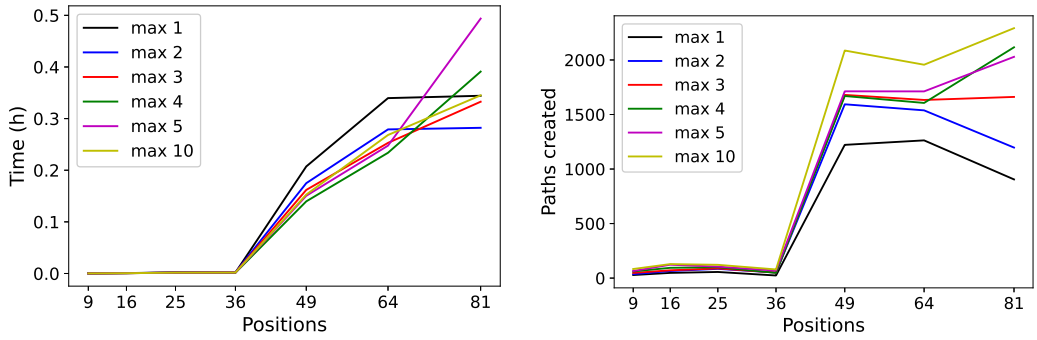


Figure 4.8: Comparison between the solutions of the problem when considering the compact mixed-integer formulation (MILP) and the column generation using either the linear program pricing problem (CGLP) or the shortest path pricing problem (CGSP).

pricing problem per iteration. Therefore, it increases the maximum number of paths created per iteration and adds many more paths to the master problem, as shown in Figure 4.9(b). As a result, we need a longer time to solve the master problem per iteration.



(a) Impact on solution time.

(b) Impact on amount of variables needed to reach final solution.

Figure 4.9: Analysis of the impact of changing the number of new variables (paths) added to the master problem per iteration of the column generation.

4.3 Conclusion

In this chapter, we presented a mixed-integer linear model to optimally deploy a FANET proposed in our first work (Dias Da Silva & Caillouet, 2020). The drones provide a connected path between the base station and each mobile sensor at all time steps. We minimised the total travelled distance of the drones and energy consumption. Moreover, we showed a clear trade-off between these two

metrics. However, the scalability of the model was limited, and we could not consider more than 25 positions.

Then, we presented a new model based on column generation with a path formulation (Dias Da Silva et al., 2021). The model can build a FANET covering mobile rescue teams deployed in a devastated area while providing a connected path to the base station for quick data collection and processing for efficient disaster management. The objective is to minimise the drones' total travelled distance, reducing energy consumption (but not minimising it) and improving communication protocols performance.

We proposed two pricing models in the column generation process, and we showed that replacing the linear program pricing problem with the shortest path problem that we can in polynomial time drastically improves the resolution time. We showed that both column generation models can scale better, and although they cannot always reach the optimal solution, their average optimality gap is no more than 5%.

With the column generations models, we can consider more positions, which increases the complexity of the problem but allows us to place the drones more precisely. Therefore, by considering more positions, we can find better solutions with the column generation than we could with the mixed-integer linear program considering fewer positions.

We explored changing the maximum number of paths created per iteration of the column generation. With few positions, more paths per iteration lead us to the solution faster. Still, with more positions, this leads to many unnecessary paths, resulting in a longer time to find the answer. This raises interest in an adaptive maximum number of paths per iteration that could lead us to the solution faster by avoiding unnecessary paths.

This upstream work is dedicated to optimising the planning of rescue teams' operations when a disaster occurs. Our models still need a lot of time to find a near-optimal solution. They also require knowing the trajectory of the sensors in advance, but we should expect these trajectories to change in these situations. Therefore it is in our interest to investigate further and study methods to update this plan during the disaster scenario to maintain coverage and connectivity as the situation evolves.

PART

Charging fixed ground sensors with drones

CHAPTER 5

Related works to drone deployment for power harvesting

The second application we studied is the deployment of drones to recharge sensors wirelessly, and this chapter presents the related works. While electronics manufacturers have made great strides in optimising the power consumption of wireless sensor batteries, regular recharging is paramount to ensure the systems' longevity and usability. Energy harvesting technology allows us to recharge batteries automatically. We can use the sun, wind, vibrations, and other natural energy sources to recharge the batteries. However, these sources are inconsistent and not always available.

Therefore, we focus on wireless Radio Frequency (RF) power harvesting such as De Donno et al. (2014). The unquestionable advantages of these solutions are flexibility, adaptability and automation. Furthermore, we can use mobile vehicles to visit the sensors regularly to recharge them and collect their data. Fleets of drones are an ideal candidate for this type of recharging. They can transmit RF signals to the sensors to recharge their batteries. The objective is to define automatic and autonomous flight plans for a fleet of drones to power up energy-constrained sensor nodes. We minimise the time required to recharge the entire system.

Using drones as chargers to enhance wireless sensor network lifetime has recently gained attention. Some works optimise the drones' trajectory, either minimising their total time or energy consumption, while others focus on resource allocation, such as maximising the energy harvested by the sensors. However, they often limit their study to deploying only one drone in the sensor network. We seek to deploy multiple drones to share the workload and reduce the time required to recharge the sensors. While a blind share of tasks among the drones can already be beneficial, we look for the optimal fleet organisation.

5.1 Energy harvesting model

We need to estimate how much energy a sensor harvests when a drone sends power from the sky. Hence, we adopt the model of Caillouet, Razafindralambo, and Zorbas (2019). We use drones as chargers with a directional antenna facing the ground to emit energy to the network and recharge the sensors continuously. A drone can adjust its altitude to recharge the batteries of multiple sensors simultaneously. Each sensor node has an RF-power harvesting module capable of converting the RF power from transmitted signals to DC power.

However, the efficiency of this conversion is not 100 %. The energy harvesting efficiency is affected by several parameters, such as environmental conditions and the distance between the source (drone) and the sensor. Zorbas, Raveneau, and Ghamri-Doudane (2016) models the amount of power harvested by a sensor s located within the line of sight of the charger u as follows:

$$P_{h_s} = P_{\text{rx}}^{d_{us}} f^{d_{us}}, \tag{5.1}$$

In Equation (5.1), $P_{\text{rx}}^{d_{us}}$ is the received power and $f^{d_{us}}$ is the efficiency of the harvesting antenna at distance d_{us}. We also use the following propagation model defined by Win, Pinto, and Shepp (2009) in Equation (5.2) to define the received power from a source at a distance d:

$$P_{\text{rx}}^{d} = P_0 \frac{e^{2\sigma G}}{d^{2b}}, \tag{5.2}$$

The term $\frac{e^{2\sigma G}}{d^{2b}}$ has a log-normal distribution with a shadowing coefficient σ ($G \sim N(0,1)$) and b is the amplitude loss exponent. P_0 is the received power at the reference distance of 1m.

The minimum harvested power a sensor receives to recharge its battery depends on the converter's efficiency for the corresponding received power at this particular distance. We denote by Γ this minimum harvesting threshold, which is a hardware-depended constant. Formula (5.3) defines the harvested energy of a sensor s for a given time period t:

$$H_s^t = \int_0^t P_{h_s} t dt. \tag{5.3}$$

This energy is stored directly in a super-capacitor with some leakage properties expressed as ηH_s^t, where $\eta \in (0, 1)$.

5.2 Optimal deployment of chargers for energy harvesting

Zhang, Qian, Kong, Wu, and Lu (2015) model the problem of optimally positioning charging devices to maximise the charging quality of sensors. Given a set of possible positions, they look for the optimal subset in which they place wireless charging devices. Moreover, they determine the transmission power of each charger. A higher transmission power means a higher range and charging quality but with an energy cost. They assume sensors can receive power from multiple chargers simultaneously without relevant losses. The authors show that the problem is NP-hard and propose heuristics to solve it. Using drones to recharge the sensors has advantages such as cost and flexibility but also adds to the complexity of the problem. We must place the chargers optimally and find their optimal trajectories.

5.3 Optimal resource allocation for drone-assisted networks with energy harvesting

Saif et al. (2021) proposed a drone system for a postdisaster scenario. As the network infrastructure is damaged and/or overloaded, devices form an ad-hoc network to keep communication, and a drone sends power to these devices to increase the system's lifetime. However, not all devices are within range, so they elect cluster heads. The cluster heads route power to devices out of the drone's range. They propose a model to elect the optimal cluster heads to increase the system's efficiency. They consider the drone can charge multiple devices simultaneously, but their model uses a single drone, and they do not model its trajectory. Moreover, they consider a powerful drone that can send power to the devices from an altitude of up to $250\mathrm{m}$. We consider more limited drones and devices requiring closer proximity for power harvesting. In our scenarios, the drones fly at most $5\ \mathrm{m}$ above the ground and can only charge devices within a $10\ \mathrm{m}$ distance (Zorbas, 2020).

Similarly, Xu, Liu, Huang, and Yuen (2021) optimises resource allocation on a drone-assisted network. They maximise user association with the drone working as a flying base station and maximise the energy harvested by them. While doing so, they minimise the delay in transmission and the drone's altitude, which ranges between $100\mathrm{m}$ and $250\mathrm{m}$. However, as in the previous work, they also consider a single drone and do not model its trajectory.

5.4 Optimal trajectory for data collection and energy harvesting

Yang, Xu, and Shikh-Bahaei (2020) propose a model for drone trajectory optimisation. A rotary-wing drone collects data from a group K of users (devices) and recharges their batteries in their

system. They consider two modes of communication. In the half-duplex mode, the users first harvest the energy and then send data to the drone. In the full-duplex mode, the users send the data while harvesting energy. Their model determines how the drone must visit all users and return to the initial position optimally.

The drone flies a straight line for each user to reach a position on top of the user at a height h. Then it gets lower to communicate with the user better. It communicates with the user for a few seconds, goes up again to h and flies straight to the following user. The authors assume h to be the height at which the drone can fly without colliding with other drones or buildings. They estimate the flight energy consumption with the model proposed by Filippone (2006). They ignore the energy consumption due to communication as it is negligible compared to the flight energy consumption.

They propose a model to minimise the system's total energy consumption while ensuring that the drone collects a minimum amount of data and that the devices harvest more energy than they spend to communicate with the drone. They divide the problem into two parts. First, they find the path with minimum energy consumption. Then, given the fixed route, they minimise energy consumption during communication. This is not the energy required to communicate (which they ignore) but the energy required to fly during communication time. It includes going from h to h_c, hovering during the communication and energy harvesting period, and returning to h.

For the path planning, they formulate an integer linear program similar to what we proposed in Dias Da Silva and Caillouet (2020) and Dias Da Silva et al. (2021), but considering a single drone. Given the high complexity, they apply a Lagrange dual method to solve the problem.

Then, their second problem is to choose the height h_c at which the drone will communicate with each user. They minimise the energy consumption for going from h to h_c and back to h. They also minimise the energy consumption for communication during the seconds the drone is at h_c. However, they must ensure the drone collects a minimum amount of data and that the devices harvest more energy than they spend to communicate with the drone. This is an interesting model, and the authors conclude that using a half-duplex mode is better for energy consumption. Although we can adapt this model to minimise the time required to recharge the sensors, which is our goal, their model considers only one drone. Moreover, in their scenario, the users are distant from each other, and they cannot leverage charging multiple sensors simultaneously with the same drone.

Arabi, Sabir, Elbiaze, and Sadik (2018) look at the same problem differently. They consider fixed sensors scattered randomly in an area and a single drone acting as the base station. The sensors form clusters, and the drone must visit all of them. The drone is equipped with an energy harvesting module, and they study the trade-off between maximising the energy given to the sensors and maximising the data collected from them. They define the optimal drone trajectory as the shortest one. Given the coordinates of each cluster, they obtain the optimal trajectory by solving the travelling salesman problem with time windows.

They evaluate different policies for deciding the priority of each device, where devices with higher priority are visited first for energy harvesting and data collection. They concluded that prioritising devices with the lowest battery level was the best approach for the network's lifetime. They also show the drone's impact on the network's lifetime grows as the network gets denser. This is expected, as the drone spends less time flying between devices and more time sending power.

Feng et al. (2020) also optimises a drone's trajectory to charge a set of sensors. They decompose the problem into sub-problems solved separately. First, they elect the optimal 2D positions the drone must visit. Then, they choose a height for each position that optimises the time to charge

the sensors. And lastly, they model the trajectory as the travelling salesman problem and use a branch and bound algorithm to solve it devised by Singh and van Oudheusden (1997).

Caillouet, Razafindralambo, and Zorbas (2019) presented a mixed-integer linear program to determine the optimal deployment of drones to recharge fixed ground sensors' batteries. They consider a discrete set of positions from which they select a subset that the drones must visit. They assign each selected position to a drone and determine the time the drone must hover it. During this time, the drone sends power to a set of sensors in a range of its position. The authors considered that multiple sensor nodes could harvest energy from one drone if they are within its coverage area. Moreover, they assume that a sensor can harvest energy from various drones simultaneously without considerable loss. They share the workload of charging the sensors among the drones. Each drone requires a total time to recharge the sensors it covers, and the model minimises the worst case. Hence, ensuring a fair division. In addition, they enforce that each sensor receives at least a minimum amount of energy denoted as its energy requirement,

5.5 Limitations

These works either consider a single drone or do not model the drones' trajectories. Moreover, Altinel and Karabulut Kurt (2016) observed that the received power depends on the number of RF sources and channel conditions. And they show, contrarily to many of the works above, that the amount of energy harvested by a node does not equal the sum of harvesting powers received by all the available sources due to spatial diversity. It is more reasonable to limit the number of simultaneous RF sources to 1 to ensure the good behaviour of the energy harvesting capabilities for the sensor nodes. But this assumption forces us to be careful in the trajectory design of multiple drones to avoid conflicts for RF transmissions.

CHAPTER 6

Charging ground sensors with drones

This chapter presents the work published in Dias Da Silva et al. (2022) and further improvements. We propose a model for a self-organised deployment of a fleet of drones to charge a set of fixed ground sensors. In our model, each sensor has an energy requirement that must be satisfied. Every sensor must receive at least its energy requirement at the end of the charging process. We consider a drone can charge multiple sensors simultaneously. Sensors can also receive power from various sources at the same time. However, as discussed in the previous chapter, this produces a loss of efficiency. Therefore, we treat such cases as conflicts and avoid them to ensure harvesting efficiency.

Finding the optimal solution to this problem is very hard. Zhang et al. (2015) showed that choosing where to place the chargers optimally is NP-hard. Our problem is a dynamic version of theirs, where the drones (chargers) move, and we want the optimal trajectories while avoiding conflicts. Given this complexity, we propose a two-step optimisation framework to minimise the recharge time of a complete sensor system. We also minimise the number of drones required and the overall flight time. We decompose the problem into two sub-problems. The first one determines deployment positions and associated charging times to meet the energy requirements of each sensor. The second one computes the optimal trajectories for the drones while avoiding conflicts. We propose a mixed-integer linear programming model for the first sub-problem. Then, we develop heuristic algorithms for the second sub-problem. We evaluate our approach on a set of randomly generated instances.

6.1 Linear Programs

In this section, we describe the linear programs responsible for the first step of our approach: determining the positions the drones must visit along with the duration of these visits such that every sensor receives its energy requirement. We propose two mixed-integer linear programs, which we call drone-based and sensor-based linear programs.

6.1.1 Drone-Based linear program (DB-LP)

This mixed-integer linear program seeks to determine the optimal number of drones needed to recharge the sensors in the minimum amount of time. We select the positions they must visit for each drone and their associated hovering time to satisfy the sensors' energy requirements.

We discretise the monitored area and derive a set P of possible 3D positions. Given the energy needs E_s of each sensor $s \in S$ and the minimum harvesting threshold Γ defined in the energy model of Section 5.1, we associate to each sensor s a set of possible positions P_s. These positions are within a 10 m distance from s, which is close enough to satisfy the threshold Γ for s (Zorbas, 2020). We define the following binary variables for drone positioning:

- $x_u = 1$ if the drone $u \in U$ is used, 0 otherwise.
- $y_p^u = 1$ if the drone u is deployed at position $p \in P$, 0 otherwise.

We also need to define how long the drones spend in a position to recharge at least the energy requirement of each sensor. We define a set t_p^u of real variables representing the time duration of drone u at position p. We also set τ the maximum total recharge time of a drone (its maximum flying time).

We want to minimise the number of used drones to recharge the sensors in a minimum amount of time to ensure the required harvested energy E_s. However, the fewer drones we use, the more each drone must work to charge all sensors. Hence, there is a trade-off between minimising the number of used drones and minimising the total recharge time. Indeed, having enough drones brings us to deploy one drone for one sensor, lasting enough time to recharge all E_s entirely. On the contrary, limiting the number of positions per drone, therefore, splits the harvested time between all the available drones, and they all go to the same positions but for a limited duration such that they will all charge a fraction of the amount of E_s and exchange positions. These two antagonistic objectives lead us to develop a bi-objective optimisation problem in which we want to limit the total recharge period and the number of positions:

$$\text{minimise } \max_{u \in U} \sum_{p \in P} t_p^u \tag{6.1}$$

$$\text{minimise } \max_{u \in U} \sum_{p \in P} y_p^u \tag{6.2}$$

In Objective Function (6.1), we minimise the time needed to harvest and store the required quantity of energy from the deployed drones to the sensors, and in Objective Function (6.2), we minimise the number of visited positions. We want to study the balance between the number of visited locations and the total flying time of the drones to derive satisfying solutions.

Since we want to compute $\min \max$, we define two continuous variables modelling the worst time λ_1 and the worst number of positions λ_2. We can combine the bi-objective into only one function $\min \alpha \cdot \lambda_1 + (1 - \alpha) \cdot \lambda_2$, where α is an input parameter to balance the impact of the two objective functions on the final solution.

$$\sum_{p \in P} t_p^u \leq \lambda_1, \quad \forall u \in U \tag{6.3}$$

$$\sum_{p \in P} y_p^u \leq \lambda_2, \quad \forall u \in U \tag{6.4}$$

We consider a B amount of available drones. Therefore, in Constraint (6.5), the number of used drones must be less than or equal to B:

$$\sum_{u \in U} x_u \leq B \tag{6.5}$$

In Constraints (6.6), the total harvested energy of each sensor s must at least its energy requirement E_s:

$$\sum_{u \in U} \sum_{p \in P_i} (1 - \eta) H_s t_p^u \geq E_s, \quad \forall s \in S \tag{6.6}$$

Then, we must verify the possibility of a drone moving around. We can place each drone at different locations during the operation, but the total length of its flying time cannot exceed the imposed time limit (i.e., τ) because of Constraints (6.7). Moreover, Constraints (6.8) ensure that if a drone is not placed anywhere, it is not used. And vice-versa, Constraints (6.9) ensure that if the drone is not used, it cannot be assigned to any position.

$$\sum_{p \in P} t_p^u x_u \leq \tau, \quad \forall u \in U \tag{6.7}$$

$$x_u \leq \sum_{p \in P} y_p^u, \quad \forall u \in U, p \in P \tag{6.8}$$

$$y_p^u \leq x_u, \quad \forall u \in U, p \in P \tag{6.9}$$

We do not place drones above other drones to avoid collision and interference. So we limit the total time for positions in the same z-axis to λ_1. This ensures if ever two or more drones visit positions in the same z-axis, we can schedule these visits to happen consecutively in less than λ_1 time:

$$\sum_{u \in U} t_p^u + \sum_{\substack{p' \in P \\ p_{xy} = p'_{xy}}} \sum_{u' \in U} t_{p'}^{u'} \leq \lambda_1, \quad \forall p \in P \tag{6.10}$$

In the scheduling phase, we do not allow multiple drones to recharge a sensor simultaneously to ensure harvesting efficiency. Therefore we avoid sharing this charging task among too many drones. To do so, we limit the total time for positions P_s in the harvesting range for sensor s to the maximum trajectory time of a drone λ_1:

$$\sum_{u \in U} \sum_{p \in P_s} t_p^u \leq \lambda_1, \quad \forall s \in S \tag{6.11}$$

6.1.2 Sensor-based linear program (SB-LP)

The linear program presented here is a simplified version of the DB-LP. The goal is to elect the positions necessary to recharge the sensors while minimising the total time to recharge each sensor as well as the number of positions used. The output of this linear program is a set of positions that the drones must visit and the corresponding visitation times. However, unlike DB-LP, SB-LP does not determine which positions each drone must visit. Then, the scheduling algorithm becomes responsible for not only deciding the order in which the positions are visited but also which drones should visit which position. This approach's primary goal is to give the scheduling algorithm more freedom to avoid conflicts reducing waiting times. The variables for the SB-LP are:

- y_p is a binary variable that is 1 if the position $p \in P$ is visited and 0 otherwise.
- t_p is a real variable representing how long a drone must stay in position $p \in P$.

And the mixed-integer linear program SB-LP is defined as follows:

$$\text{minimise} \quad \alpha\lambda_1 + (1-\alpha)\lambda_2 \tag{6.12}$$

$$\sum_{p \in P_s} t_p \leq \lambda_1 \quad \forall s \in S \tag{6.13}$$

$$\sum_{p \in P_s} y_p \leq \lambda_2 \quad \forall s \in S \tag{6.14}$$

$$\sum_{p \in P_s} (1-\eta) H_s t_p \geq E_s \quad \forall s \in S \tag{6.15}$$

$$t_p \leq y_p \tau \quad \forall p \in P \tag{6.16}$$

Constraints (6.13) limit the maximum time to recharge a sensor to λ_1. For a sensor s, P_s is the set of positions that are close enough to transmit a non-negligible amount of power to s. The sum of the time spent in each position $p \in P_s$ is the total time spent charging s. Similarly, Constraints (6.14) limit the maximum number of positions used to recharge a sensor to λ_2.

We must still ensure that every sensor will be recharged by solving this mixed-integer linear program. This is done by Constraints (6.15). Finally, Constraints (6.16) ensure the definition of the variables t_p and y_p. If we associate a time bigger than 0 to a position p, then y_p must be one. So we minimise the total time to charge each sensor and the number of positions used. α is a given parameter that balances these two metrics' trade-offs in Objective Function (6.12).

6.2 Scheduling Algorithms

The linear models presented in Section 6.1 give as output the positions that the drones must visit and the time spent in these positions. The next step is to compute the full trajectory of each drone, visiting successively these positions. For this, we define the output of the linear models as a set of tasks, where a task k is given by a position p_k, a duration t_k in seconds, and a set of sensors S_k that is recharged during the task's execution as in Figure 6.1. For the DB-LP model, each task k also has a drone u_k assigned, while the SB-LP model does not provide this information. Each drone can have multiple tasks to complete, and our goal here is to determine the order in which the drones complete their respective tasks without *conflicts*.

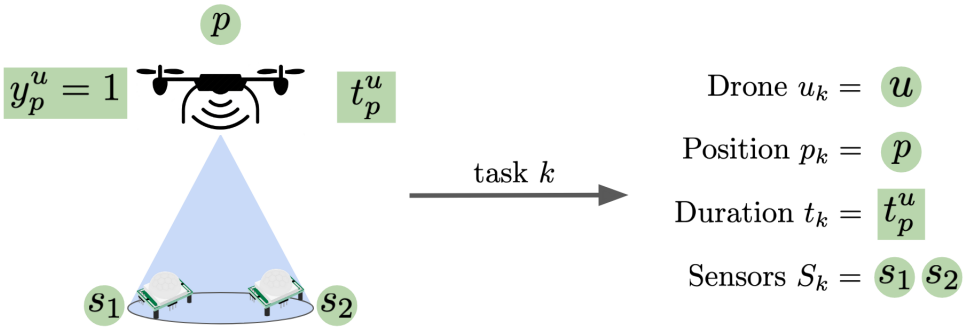


Figure 6.1: Task example for DB-LP.

There is a *conflict* between two tasks k and k^* if:

1. They use the same drone ($u_k = u_{k^*}$).
2. They use the same position ($p_k = p_{k^*}$).
3. They charge some common sensors ($S_k \cap S_{k^*} \neq \emptyset$) (to avoid the multi-charger constraints of Altinel and Karabulut Kurt (2016)).

A task is said to be *conflict-free* at a given moment if it has no conflict with any currently executed tasks. We define $C(k)$ as the set of current tasks that conflict with k, and therefore, we denote a conflict-free task by $C(k) = \emptyset$.

Even if we ignore the possible conflicts between tasks, determining the fastest order of tasks for each drone is the same as solving the Traveling Salesman Problem (TSP), an NP-hard problem. We, therefore, present greedy algorithms that assign tasks to minimise the total time the drones have to wait to avoid conflicts. Just as we defined the linear programs as "drone-based" and "sensor-based", the scheduling algorithms are:

- **Drone-Based algorithms:** These algorithms receive a set of tasks as their input, where each task already has a drone assigned to it (such as the output of the DB-LP model). They decide the order in which the drones will execute their tasks while avoiding conflicts.
- **Sensor-based algorithms:** These algorithms receive a set of tasks as their input, where the tasks do not have drones already assigned to them (such as the output of the SB-LP model). They decide which drones will be responsible for each task and the order of execution. We can use the output of the DB-LP as input for these algorithms. In that case, they will ignore the drone assignment from the linear program.

Table 6.1: Algorithms' Variables Definition

Variable	Definition
K	Input set of tasks
O	Output set of tasks
U	Set of available drones
t	Current time
u_k	Drone responsible for task k
s_k	Start time of task k
t_k	Duration of task k
f_k	Finish time of task k
p_k	Position of task k
i	Amount of tasks not in the set O
$\mathrm{Task}(u)$	Last task assigned to drone u
$\mathrm{Position}(u)$	Last position of drone u
$C(k)$	Set of tasks currently assigned to drones and in conflict with k
$\mathrm{Flight}(k)$	Amount of time required for u_k to reach p_k
$\mathrm{WaitTime}(k)$	Amount of time before the task k can be executed conflict-free

Table 6.1 defines all variables used by the algorithms. We define $\mathrm{Flight}(k)$ as the time required by the drone u_k to reach the position of the task k. If u_k is free, $\mathrm{Task}(u_k) = \emptyset$, then $\mathrm{Flight}(k)$

is the distance between the two positions divided by the drone's speed. If the drone has a current task $k*$, then we add the time remaining for this task to finish before the drone can fly, $f_{k^*} - t$:

$$\text{Flight}(k) = \begin{cases} \text{distance}(p_k, \text{Position}(u_k))/\text{speed}, \\ \quad \text{if Task}(u_k) = \emptyset \\ \\ \text{distance}(p_k, \text{Position}(u_k))/\text{speed} + f_{k^*} - t, \\ \quad \text{if Task}(u_k) = k^* \end{cases} \tag{6.17}$$

The $\text{WaitTime}(k)$ is the soonest a task can start being executed without conflicts. If there are no conflicts, we only need to wait for the time $\text{Flight}(k)$ for the drone to arrive at the task's position. If there are conflicts, but all tasks in conflict with k will finish before the drone even arrives at p_k, then we only need to wait for the time of flight. The conflicts are only a problem if the time remaining for them to finish is greater than the time to reach the position p_k. In this case, we have no choice but to wait for the conflicts to finish before starting the task's execution. The $\text{WaitTime}(k)$ is defined as:

$$\begin{aligned} \text{WaitTime}(k) &= \max(\text{LastConflict}(k), \text{Flight}(k)) \\ \text{LastConflict}(k) &= \max(f_{k^*}, \forall k^* \in C(k)) \end{aligned} \tag{6.18}$$

Whenever a drone has to wait for a conflict resolution before executing a task, we assume a delayed flight such that the drone arrives at p_k precisely as the last conflict ends. Otherwise, we consider the drone leaves its current position immediately.

6.2.1 Drone-Based Shortest Tasks First (DB-STF)

Algorithm 6.1 implements DB-STF. It receives a list of tasks K from the DB-LP model as input. Meaning it needs to have the drones already assigned to the tasks. Its output is a set O with the same tasks with specified start and finish times and, therefore, scheduled. The algorithm determines the tasks' scheduling priority by their duration time. The shortest tasks have higher priority.

First, the variables are initialised. The current time t is set to 0, and no drone u has a task assigned to it yet. Likewise, each drone's initial position is the base station. The output set O starts as an empty set, and the number of tasks i that do not belong to O is $|K|$. Once these variables are initialised, we order all tasks in ascending order according to their duration time.

After the initialisation finishes in line 8, the algorithm begins its main loop with three steps. These steps are the tasks' assignment, the time jump and the tasks' conclusion. The main loop repeats these steps until all tasks are executed, i.e. while there is any task not in the output set $(i > 0)$.

The first step, the tasks' assignment, happens from line 12 to line 21. It only happens when there are still tasks in K. In this step, we check if there is any task in $k \in K$ whose drone has no task currently assigned to it ($\text{Task}(u_k) = \emptyset$). If there is no such task, no assignment happens. However, if at least one such task exists, we choose the one that appears first in K (highest priority) and assign it to its corresponding drone. This assignment defines its start and finish times based on the wait time. We also need to update the drone's position to the position of k and the drone's

current task value. Once k is assigned, if there are still tasks in k, we repeat this step, but if there are no more tasks in k or no tasks whose drones are free, we move to the next step.

Note that the assigned task does not necessarily have the highest priority in K. It is just the task with the highest priority among the ones whose drones are free. It is also important to note that we must calculate the wait time again for the remaining tasks after each assignment. This happens because the new task changes its drone's position, which affects the flight time of other tasks that depend on the same drone. Additionally, it might conflict with some of the remaining tasks because of a common position or target sensor. Therefore, the change in flight time and conflicts impact the wait time for the remaining tasks.

The second step is simply updating the current time as shown in line 24. Among all the tasks currently assigned to the drones, we choose the shortest finish time and update our current time to its value.

The third step verifies which tasks have been finished. Only one task is expected to finish at each main loop iteration. But more than one task might end precisely at the same time. Therefore, we verify among the current tasks which ones have a finish time equal to or smaller than the present time. All these are added to the output set, and we mark their drones as free.

We repeat these three steps until the current time is equal to the finish time of the last task. All tasks are in the output set O, meaning all have their start and finish times defined, and the algorithm finishes.

As expected, the drone-based longest tasks first algorithm (DB-LTF) is the same except for the ordering that is in descending order.

6.2.2 Drone-Based Time of Flight (DB-TOF)

The drone-based time of flight algorithm, shown in Algorithm A.1, prioritises the tasks by how much time is needed to reach their position. The earlier drone u_k can get to position p_k, the higher the scheduling priority of k. DB-TOF receives as input a list of tasks with predefined drones, such as the output of DB-LP. The variables' initial values in the algorithm do not change compared to Algorithm 6.1.

There are two main differences between DB-TOF and Algorithm 6.1. First, whenever the current tasks change, the priority of the remaining tasks changes. This happens because the drones move, impacting the time to reach the remaining tasks' positions. Hence, the ordering of the tasks happens at the assignment step instead of single ordering at the beginning of the algorithm.

The second difference is that we only assign tasks that have no conflicts with the current tasks. Therefore, even though each assignment can impact the flight time of the remaining tasks, we do not need to order K after each assignment because we do not consider tasks with conflicts.

Once again, the tasks assigned are not necessarily the one with the highest priority in K, but the ones with the highest priority such that their drone is currently free and have no conflicts with any current task.

The two last steps remain the same. Update the current time to the shortest finish time, add all finished tasks to the output set and mark their drones as free.

6.2.3 Drone-Based Wait Time (DB-WT)

This algorithm, shown as Algorithm A.2, is also drone-based and requires drones already assigned to their corresponding tasks in the input. Thus, we use the output of the DB-LP model to create

Input: Lists of tasks K such that $u_k \neq \emptyset, \forall k \in K$
Output: A list of tasks O with all tasks and their start and finish time, denoted s_k and f_k respectively
$t = 0$
$\text{Task}(u) = \emptyset, \forall u \in U$
$\text{Position}(u) = \text{base\_station}, \forall u \in U$
$i = |K|$
$O = \emptyset$
Order K by ascending t_k
while $i > 0$ **do**
  $\text{assign\_tasks} = (|K| > 0)$
  while assign_tasks **do**
    assign_tasks = *FALSE*
    Determine $\text{WaitTime}(k)\ \forall k \in K$
    $k = \text{first}(k \in K$ such that $\text{Task}(u_k) = \emptyset)$
    if $k \neq \emptyset$ **then**
      $s_k = t + \text{WaitTime}(k)$
      $f_k = s_k + t_k$
      $\text{Position}(u_k) = p_k$
      $\text{Task}(u_k) = k$
      Remove k from K
      $\text{assign\_tasks} = (|K| > 0)$
    end if
  end while
  $t = \min\{f_k$, such that $k = \text{Task}(u), \forall u \in U\}$
  for all $n \in N$ **do**
    $k = \text{Task}(u)$
    if $f_k \leq t$ **then**
      Add k to O
      $\text{Task}(u) = \emptyset$
      $i = i - 1$
    end if
  end for
end while
return O

Algorithm 6.1: Drone-Based Shortest Tasks First (DB-STF)

the set of tasks K. Here the scheduling highest priority is given to the tasks with the shortest wait time.

There are two main differences between DB-WT and DB-STF (Algorithm 6.1). First, the tasks' scheduling priority changes with every assignment since their wait time can be impacted by the new current task. Second, we always either assign the task with the highest priority or wait until it can be assigned.

Hence, we see that the initialised values do not change, but the ordering of the tasks now happens during the assignment step. In the first step, we determine the wait time for each task and choose the task with the minimum one. Then, either this task, whose priority is the highest, has its drone free and is assigned, or we move to the next step. If the task is assigned, we repeat this first step. We reevaluate the tasks' wait time to find the new one with the highest priority and verify if we can assign it.

The last two steps are the same. We jump to the shortest finish time among the current tasks, then add this task to the output set along with any others that might have finished as well.

Figure 6.2 illustrates the Wait Time algorithm. The linear program has computed 12 tasks assigned to 4 drones to cover 40 sensors. The lines represent the time a drone spends flying, and the rectangles represent the period of execution of the tasks when the drones send RF signals to recharge the sensor batteries.

All drones start at the same location corresponding to a base station located at position $(0, 0)$. The tasks are ordered following the Wait Time algorithm. Task 4 has the shortest wait time (at the beginning, the wait time is simply the time of flight), so it is the first one to be assigned. Now we must reorder the remaining tasks because their wait time may change if they conflict with Task 4. For example, the wait time for Task 7 is no longer its time of flight, but f_4, the time remaining for Task 4 to finish. Tasks 0 and 11 again have the same wait time because they are conflict-free and have the same time of flight. Once Task 0 is assigned, we reorder the tasks, and as the wait time for Task 11 remains the smallest, we give it to Drone 3.

The tasks are reordered once more, and, Task 8 is the one with the shortest wait time since it only needs to wait for Task 0 to finish. Task 2 also needs to wait for Task 0 to finish, but since they use the same drone, it can only travel after Task 0 finishes, so its wait time is larger than Task 8. The execution of the algorithm continues until the chronogram shown in Figure 6.2 is obtained.

Figure 6.3 shows the schedule DB-TOF produces for the same example. The main difference is that it gives a higher priority to Task 7 since it has a shorter time of flight than Task 8. However, Task 7 conflicts with Task 4, which results in a big idle time for Drone 2.

6.2.4 Sensor-Based Shortest Tasks First (SB-STF)

Per the definition of the sensor-based algorithms, Algorithm 6.2 receives a list of tasks as inputs and does not require the tasks to already have assigned drones. In this new approach, we remove the unnecessary complexity of the previous algorithms, such as the notion of current time and tasks' conclusion.

We just need to keep track of the drones' last position and last assigned task. Each task assigned goes directly into the output set O. We order the tasks by their duration in ascending order and assign them one by one.

The main loop goes until all tasks are assigned and therefore removed from K. The first task $k \in K$ is assigned at each iteration. For this task, we choose the drone that can start executing it the soonest and set its start and finish time accordingly. We then update the drone's last position

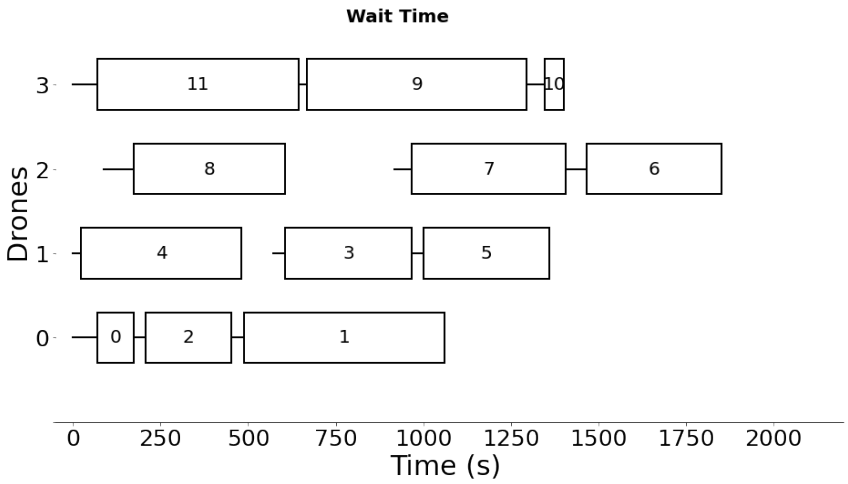


Figure 6.2: Example using DB-WT algorithm to schedule the tasks in a 5x5 grid with 40 sensors and 4 drones.

and task assigned, remove the task from K and add it to the output set. Once again, changing the ordering to descending order suffices, and this algorithm becomes the SB-LTF.

The notion of current time does not exist in this algorithm, so when computing $\mathrm{Flight}(k)$ and $\mathrm{WaitTime}(k)$, we use $t = 0$. In the drone-based version, the assignment only happened to conflict-free tasks, even if they weren't the shortest task. The sensor-based version strictly follows the order of the tasks.

6.2.5 Sensor-Based Time of Flight (SB-TOF)

For the sensor-based time of flight, shown in Algorithm A.3, we must remember that the time to reach the tasks' positions can change at each assignment. Therefore, since we are looking for the smallest $\mathrm{Flight}(k)$ in the remaining tasks K, we need to reorder the tasks at each assignment.

Hence, at each iteration of the main loop, we choose the best drone for each of the remaining tasks. This is the drone with the minimum $\mathrm{WaitTime}(k)$. In doing so, we also calculate the $\mathrm{Flight}(k)$ for all the remaining tasks. Then, we assign the one with the highest priority.

In the drone-based approach, we would only assign the task if it was conflict-free $(C(k) = \emptyset)$. Otherwise, we would consider the next task on the list. Consequently, the task with the highest priority at a given time was not necessarily assigned. Here, the sensor-based approach follows strictly the ordering of the set.

6.2.6 Sensor-Based Wait Time (SB-WT)

The sensor-based wait time algorithm, shown as Algorithm A.4, is identical to the SB-TOF save for ordering the tasks. As in SB-TOF, we need to find the best drone for each task at each assignment since the $\mathrm{Flight}(k)$ and $\mathrm{WaitTime}(k)$ of any task can change. The task with the highest priority, minimum wait time, is assigned at each iteration.

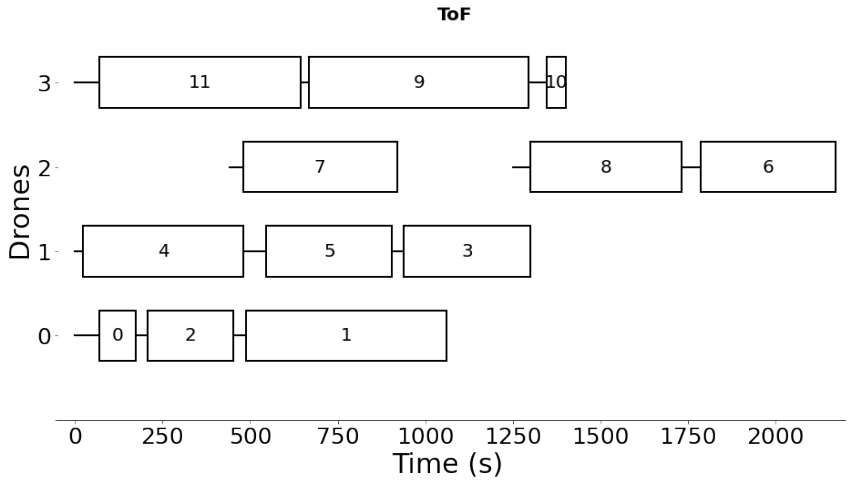


Figure 6.3: Example using DB-TOF algorithm to schedule the tasks in a 5x5 grid with 40 sensors and 4 drones.

6.3 Optimal scheduling and lower bound

We also implemented the DB-Optimal scheduling. It receives tasks from the DB-LP and looks at all possible schedules. It returns the schedule with a minimum total recharge time that avoids all conflicts. However, considering all possible permutations is computationally expensive. Hence, we propose the TSP scheduling.

The TSP scheduling is a lower bound to the optimal solution of the drone-based approach. Given a set of tasks from DB-LP, we solve the travelling salesman problem for each drone. We obtain the fastest visitation order for each drone and schedule the tasks in this order while ignoring any conflicts. Consequently, the TSP scheduling is usually invalid, but it is the optimal solution in a scenario with no conflicts. Hence, in our problem, scheduling the tasks to finish faster than in the TSP schedule is impossible. When we cannot find the optimal solution, we use the TSP schedule as a lower bound to our solutions.

6.4 Results

We consider a 2-dimensional area A of size $50\,\mathrm{m} \times 50\,\mathrm{m}$ with sensors randomly placed with uniform distribution. We divide A as a 5×5 grid with 5 possible altitudes, totalling 125 possible deployment positions for the drones to cover A. The number of sensors varies between 5 and 50, while the number of drones available to charge them varies from 3 to 10. Each sensor requires 150 mJ, and the energy model parameters are $P_0 = 10\mathrm{mW}$, $b = 1.05$, $\sigma = 1$, $\Gamma = 6.3 \times 10^{-5}$ W, and $\eta = 0.3$ (Caillouet, Razafindralambo, & Zorbas, 2019). We set the maximum recharging time of a drone to 1 hour and $\alpha = 0.1$. The linear programs were implemented in Java and solved using IBM CPLEX solver on an Intel Core i9-10900K CPU, 3.70GHz, 64 Gb RAM computer with Fedora system, release 35. We ran each linear program over 2800 different examples of sensors randomly distributed over A.

Input: Lists of tasks K
Output: A list of tasks O with all tasks and their start and finish time, s_k and f_k respectively, and drone u_k
$\text{Task}(u) = \emptyset, \forall u \in U$
$\text{Position}(u) = \text{base\_station}, \forall u \in U$
$O = \emptyset$
Order K by ascending t_k
while $|K| > 0$ **do**
    $k = \text{first}(k \in K)$
    $u_k = n$ such that n has $\min\{\text{WaitTime}(k)\}$
    $s_k = \text{WaitTime}(k)$
    $f_k = s_k + t_k$
    $\text{Position}(u_k) = p_k$
    $\text{Task}(u_k) = k$
    Remove k from K
    Add k to O
end while
return O

Algorithm 6.2: Sensor-Based Shortest Tasks First (SB-STF)

6.4.1 Linear Programs Positioning

The first step of our framework is determining the optimal positions the drones should visit and for how long to recharge the battery of the sensors using RF signals. The drone-based and the sensor-based linear programs do it differently. On the one hand, DB-LP considers the number of drones available and minimises the maximum workload of the drones from their point of view. During this process, the linear program decides which drone visits which position. On the other hand, SB-LP only considers the sensors and aims to minimise the maximum number of positions used to charge each sensor as well as the total time to charge them. However, it does not consider which drone will visit each position. It just selects a set of positions to be used and for how long they must be used. Figures 6.4 and 6.5 show how the outputs of each linear program behave as we vary the number of available drones and sensors to recharge.

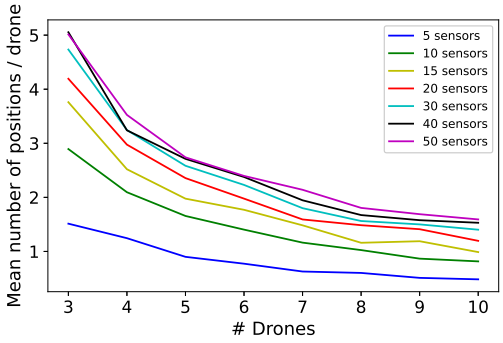
As one can observe in Figure 6.4(a), the more sensors to charge, the more positions the drones must visit. On the other hand, the more drones we have, the fewer positions they need to visit on average. The DB-LP adapts as the number of drones increases and shares the workload among them. The SB-LP, however, has the same output for any number of drones. Figure 6.5(a) shows the average number of positions selected by SB-LP divided by the number of drones we consider in the second step of our solution. However, it is up to the sensor-based scheduling algorithms to decide which positions each drone will visit. Nonetheless, the SB-LP uses fewer positions than DB-LP on average. Notice that for small amounts of sensors, if we have an excessive number of available drones, the average number of positions visited by each drone is below 1.

The altitude of the drones is also an essential factor. The energy harvesting capability is strongly related to the distance between the RF emitting source and the destination. The closer in line of sight, the better. As depicted in Figure 6.4(b), our DB-LP model seeks to place the drones as low as possible to maximise the energy the sensors harvested in the shortest time. So the lowest positions at 1 meter are the most used because of the faster battery recharge. However, higher

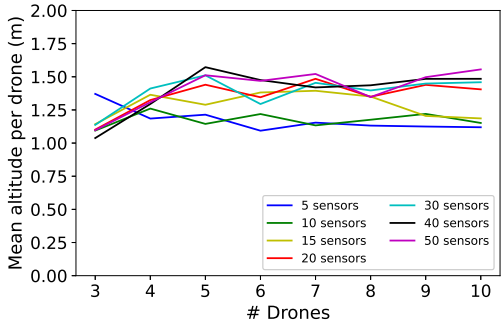
altitudes increase the drone's coverage range. The trade-off between coverage range and recharge time can be beneficial when the number of sensors increases. We can observe it in Figure 6.4(c), depicting the average time a drone spends in the same position. With a higher density of sensors, it is advantageous to cover more targets at once, even though the time to charge them increases. The same phenomenon is also observed for the SB-LP in Figures 6.5(b) and 6.5(c) that show the selected positions' average altitude and time as the number of sensors increases.

In addition, the SB-LP positions' altitudes and spent time are higher not only due to the number of sensors but also because of the smaller number of positions chosen by the SB-LP compared to DB-LP. The smaller number of selected positions by the SB-LP is compensated by their altitudes, which impacts the time spent in them.

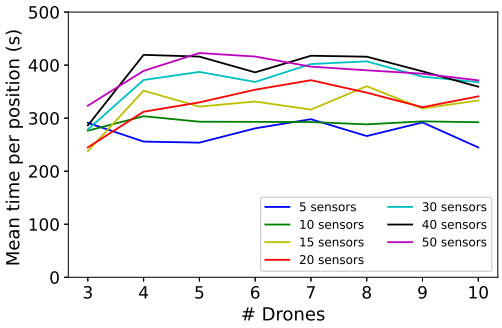
Overall, using the point of view of the drones or the sensors to reduce the total recharge time results in some differences since neither linear program considers the complete problem. These differences seem small but can have a big impact once we apply the scheduling algorithms to these tasks.



(a) The mean number of positions each drone must visit.



(b) The mean altitude of the positions occupied by drones.

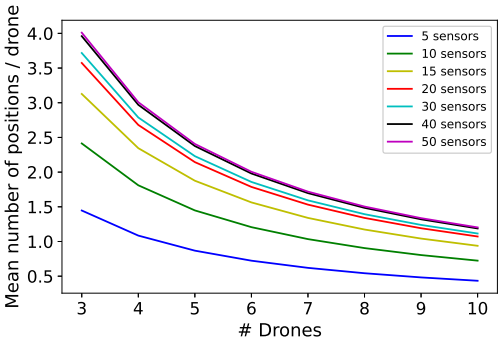


(c) The mean time each drone spends in the same position (tasks' mean duration).

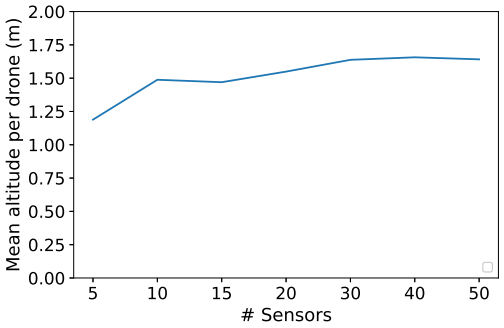
Figure 6.4: Drone-Based LP: Positioning analysis.

6.4.2 Scheduling Algorithms Performance

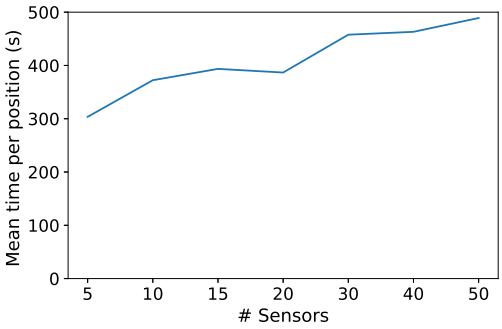
Given the tasks created by the linear programs, we must schedule them. We started this work with the notion of a drone-based and a sensor-based approach. Following these approaches, the drone-



(a) The mean number of positions each drone must visit.



(b) The mean altitude of the positions occupied by drones as the amount of sensors increases.



(c) The mean time each drone spends in the same position (tasks' mean duration) as the amount of sensors increases.

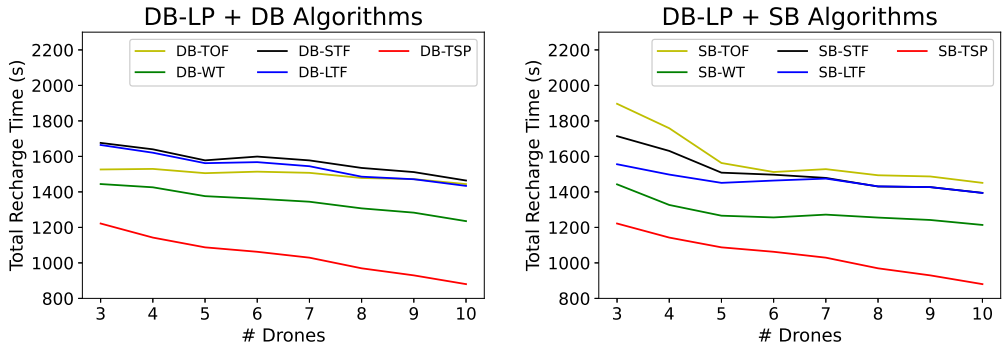
Figure 6.5: Sensor-Based LP: Positioning analysis.

based algorithms would schedule the tasks of the DB-LP, and the sensor-based algorithms would schedule the tasks of the SB-LP. However, a third approach is possible, the mixed approach. It consists of using sensor-based algorithms to schedule tasks from the DB-LP. It suffices to ignore the drone assignment done by DB-LP and only consider its chosen positions and duration of stay.

In Figure 6.6, we see the total recharge time as we change between the three approaches. We also plotted the TSP schedule for comparison using the DB-LP as input. As discussed in Section 6.3, this scheduling ignores the conflicts and is defined as a lower bound of our solutions.

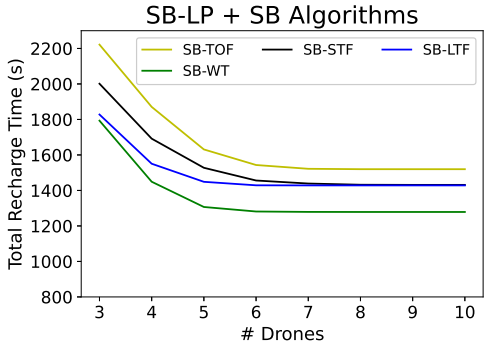
Best Scheduling Priority

Among all metrics used to define the tasks' scheduling priority, the wait time is the one that provides the best total recharge time in all approaches, as shown in Figure 6.6. The time of flight is the second best if we follow the drones' assignment by the DB-LP, but when the sensor-based algorithm assigns the drones using the time of flight as a priority, it becomes the worse metric to use. This is intuitive because this metric fails to consider the execution time of the tasks and consequently provides an uneven division of labour to the drones. The drone-based approach avoids this because the DB-LP is responsible for this division.



(a) Drone-based algorithms using drone-based LP as input.

(b) Sensor-based algorithms using drone-based LP as input.



(c) Sensor-based algorithms using sensor-based LP as input.

Figure 6.6: Total recharge time for the drone-based, sensor-based and mixed approaches: DB-LP + DB algorithms, SB-LP + SB algorithms and DB-LP + SB algorithms, respectively. Over 2800 DB-LP inputs and 2800 SB-LP inputs.

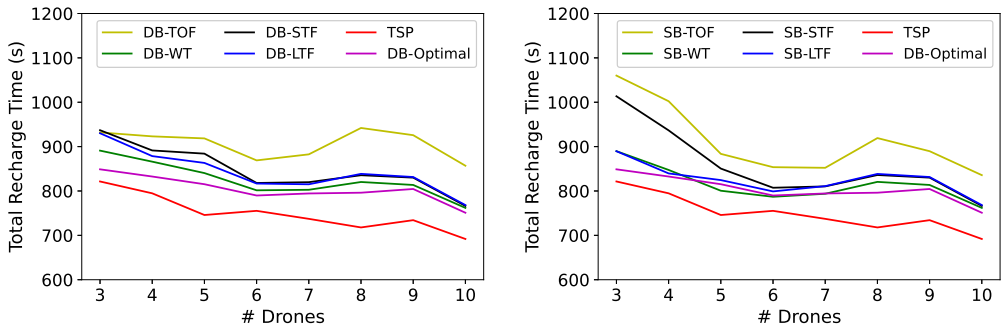
Choosing the shortest or longest tasks first is not much different in the drone-based approach, but it does favour the LFT metric. Recall that DB-STF and DB-LTF assign the task with the highest priority whose drone is free. So at a given moment, the task assigned is not necessarily the remaining task with the highest priority. Consequently, DB-LTF takes the biggest tasks, and when it cannot, it fills in with the smallest ones. However, DB-STF uses most of the shortest tasks at the beginning, and at the end, it must handle the biggest tasks simultaneously.

There is a different behaviour for the sensor-based scheduling with these metrics over the DB-LP and SB-LP. Still, the LTF metric is the best overall compared to STF, but their performances are equivalent depending on the number of drones available. This happens because the sensor-based approach strictly follows the priority order, so SB-LTF will distribute the largest tasks among the drones and fill up with the shortest tasks at the end. However, SB-STF does the opposite. It assigns the biggest tasks last, ending up with a bad division of labour when there are few drones. But when we have more than six drones, as we see in Figure 6.5(a), the amount of tasks per drone is less than 2, meaning both approaches end up with, at worst, one big task and one short task for a single

drone if the tasks' duration has a big variation. If the variation is small, then the drones have a similar workload.

The wait time is the metric with the best performance in all approaches. By definition, it considers the conflicts between the tasks, the time of flight and the duration of the tasks already assigned. Consequently, it provides a better drone assignment and scheduling of tasks. SB-WT can even find a better drone assignment than the DB-LP, whose goal is to find the best division of labour because the linear program does not consider the conflicts in its optimisation. This is shown in Figure 6.7 where we consider only 870 drone-based inputs for which we can find the DB-Optimal solution in a feasible time. The DB-Optimal algorithm is the best possible scheduling following the drone assignment of DB-LP. We see that SB-WT can reach better solutions with the mixed approach than DB-Optimal with a different assignment of tasks between the available drones.

In Figure 6.8, we see the average idle time of each scheduling algorithm with DB-LP input. The average idle time is how long, on average, the drones must wait for a task completion before executing a task assigned to them. SB-WT is more efficient than any other scheduling algorithm.



(a) Drone-based algorithms.

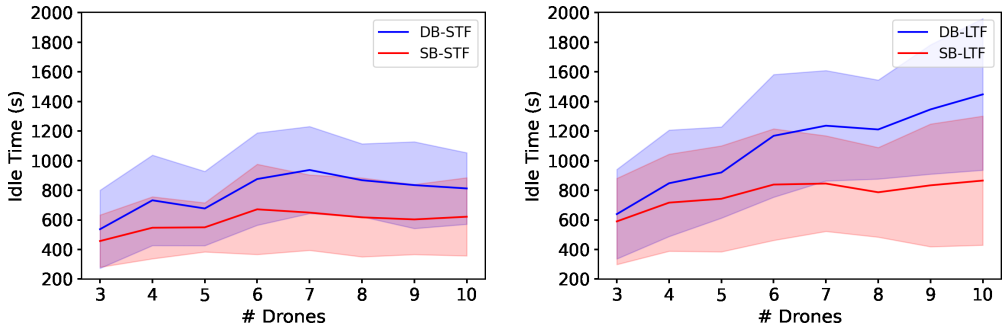
(b) Sensor-based algorithms.

Figure 6.7: Scheduling algorithms compared to TSP and DB-Optimal solution over 870 DB-LP with less than 15 sensors and less than 12 tasks.

Best approach

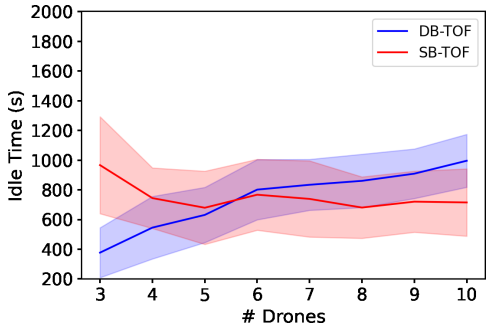
Although Figure 6.6 makes it easy to compare the priority metrics, it is not so simple to see which approach gives the best results. Hence, we show the same results differently in Figure 6.9, where comparing each approach's performance with a given priority metric is easier. We can see that the STF and LTF metrics perform better with the sensor-based approach with more than five drones, with the mixed approach close by. For the TOF, the drone-based approach is the best overall. The drone-based LP provides a better drone assignment for a small number of drones, and for more than five drones, the three approaches behave similarly.

The mixed approach provides the best results for the wait time, our best priority metric. This indicates that the sensor-based scheduling performs well, especially for this metric, but that both linear programs have limitations. Even though the DB-LP finds an optimal division of labour between the drones, ignoring this assignment with SB-WT is beneficial. This is not so unexpected since the LP does not consider conflicts. The sensor-based approach has good results for more than five drones but consistently performs worse for all metrics when we have only 3 or 4 drones.

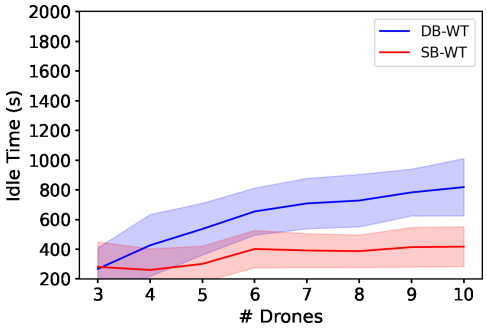


(a) DB-STF and SB-STF

(b) DB-LTF and SB-LTF



(c) DB-TOF and SB-TOF



(d) DB-WT and SB-WT

Figure 6.8: Average idle time and standard deviation for each metric comparing the DB and SB scheduling with the DB-LP input.

Overall, we consider the sensor-based approach with the wait time metric to be the best. Even though we could benefit more from the mixed approach when only 3 or 4 drones are available, using the SB-LP to create the tasks still has one crucial advantage: the solution time. SB-LP has fewer variables than DB-LP, significantly impacting the time to solve the linear program as the scenarios considered increase. Figure 6.10 shows how quickly the time to solve DB-LP increases with more sensors while SB-LP is much faster. We can solve the linear program faster for five or more drones and still reach better results.

What to improve

Notice that this framework has a significant limitation. Given how we avoid conflicts altogether, we do not benefit as much from more drones. This is very evident in Figure 6.6(c). Indeed, there exists a physical limit to the solution. Eventually, there is simply no way of charging the sensors faster than the batteries' chemical reactions allow or arriving at them faster than the drones can fly. So even if we had a perfect solution, we would eventually reach a limit as the number of available drones increases. But our solution is not perfect, and there are some things we can improve.

We avoid multiple drones charging the same sensor simultaneously because there is a loss of efficiency. Hence, the sensor does not receive the same amount of power as if the drones did it in sequence. Since we wanted the linear programs to ensure the amount of power received by each sensor, we avoided this conflict altogether. However, if we have extra drones that cannot improve

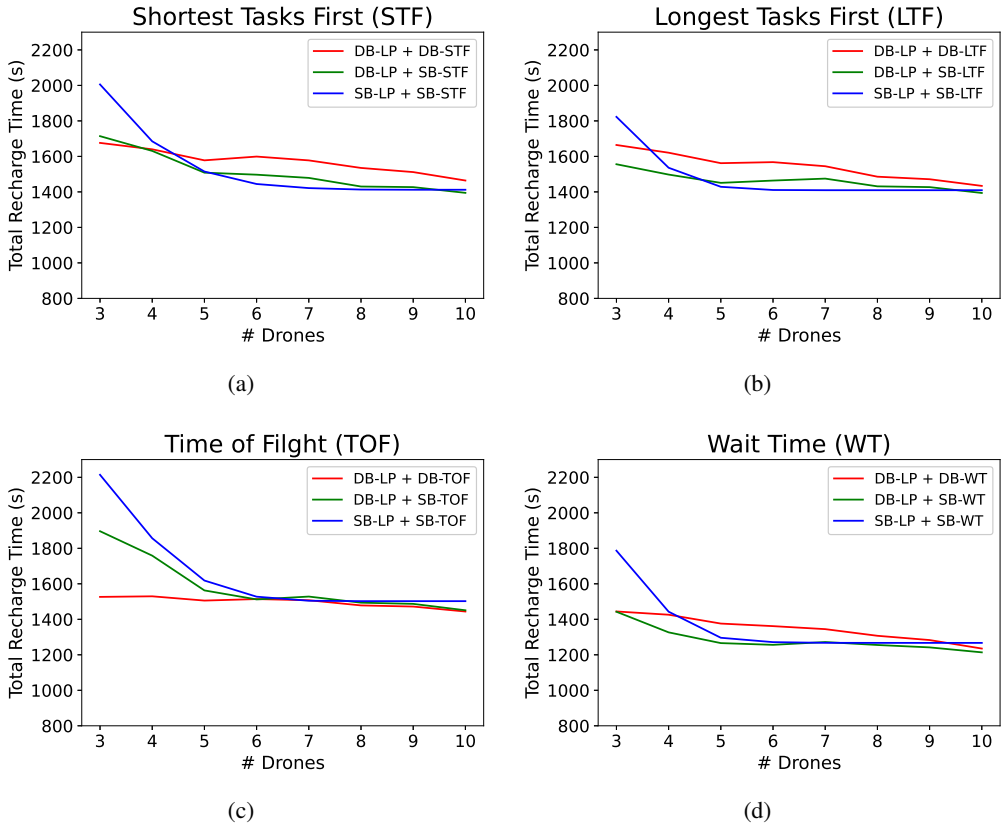


Figure 6.9: Average recharge times for each metric for each approach.

the scheduling anyway, as shown in Figure 6.6(c), then we could use them to help the other drones. Two drones sending double the amount of power will not halve the recharge time of a sensor, but they will be able to do it faster. Altinel and Karabulut Kurt (2016) show that doubling the power sources can increase the power received by the sensor between 25% and 50%. So we can use the excess drones to reduce the charging time of the sensors.

Another limitation is the complexity of solving mixed-integer linear programs. Therefore, we are exploring other approaches, such as clustering algorithms, to determine the tasks' positions and duration. Clustering heuristics can lead to better deployments as they do not depend on a predefined discrete set of positions, as shown in Mahoro Ntwari et al. (2021). Moreover, clustering can make this approach more scalable and suitable for real-size scenarios.

6.5 Conclusion

We proposed three approaches to schedule the sensors' batteries' recharge. The drone-based approach uses DB-LP to build the tasks and DB algorithms to schedule them among the drones. The sensor-based approach uses SB-LP and SB algorithms. And the mixed approach uses DB-LP and SB algorithms, which ignore the drone assignment of the linear program. We proposed four different metrics for the scheduling algorithms to decide the priorities of the tasks. Longest tasks

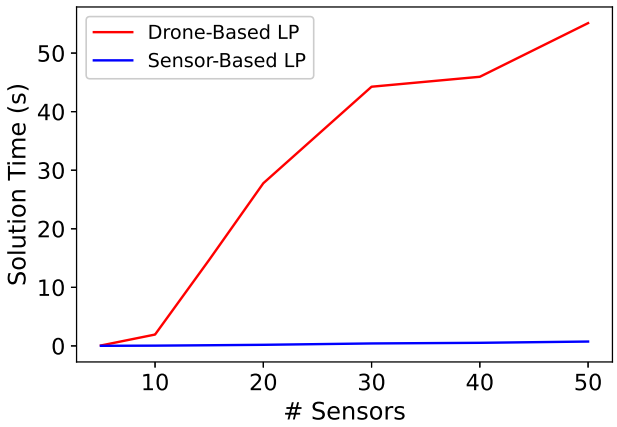


Figure 6.10: Comparison of solution time between both linear programs.

first, shortest tasks first, time of flight, and wait time. We presented the drone-based approach in Dias Da Silva et al. (2022). We have developed the sensor-based and mixed approaches since then.

The best solution is the wait time priority metric with the sensor-based approach. We could find better results with the mixed approach for a small number of available drones, but we still consider the sensor-based approach to be better, given that the SB-LP can be solved faster. For a more significant number of available drones, the sensor-based approach is faster and better.

So far, we completely avoided possible conflicts, such as reduced power harvesting efficiency. However, we believe it is possible to improve our current solution by relaxing this restriction and leveraging the increased transmission power of multiple sources. This would allow us to reduce the tasks' duration and find even better solutions. But it must be done carefully to ensure each sensor still receives at least its energy requirement. Moreover, clustering techniques to elect the deployment positions are also a promising approach to improve the solution's scalability and quality.

CHAPTER 7

Conclusion et Perspectives

In this thesis, we studied two problems. Firstly, we investigated the optimal deployment of FANETs to connect mobile sensors with the base station. Secondly, we studied the optimal deployment of drones to recharge fixed sensors wirelessly.

In the first problem, we proposed an optimal model to find the drones' trajectories. These trajectories ensure that all sensors have a communication path to the base station at each time step. We optimised the drones' total travelled distance and energy consumption. By using a more precise energy model, our model considers the trade-off between energy and distance. Moreover, we found good solutions that minimise both metrics simultaneously. However, the complexity of finding these optimal trajectories was too high.

Then, we used column generation, an optimisation method that allows us to find near-optimal solutions much faster. We also represented our pricing problem as a shortest-path problem that we can solve in polynomial time, which improved the scalability of our model even more. We were able to find solutions for instances with up to 100 positions when before, we could not consider more than 25. This higher scalability allowed us to find even better solutions where the drones move more precisely. Furthermore, our solutions are only 5% away from the optimal ones.

While our model is optimal, it is not practical. Our near-optimal model scales better but still cannot consider more significant scenarios. We could implement it in practice if we divided extensive areas into smaller ones and solved many simpler versions of our problem. However, this would not be optimal. Even in this case, we would need to do it in advance, and we would not be able to adapt to changes in the environment. Therefore, going forward, we would like to study more straightforward solutions focusing on scalability, robustness and adaptability instead of optimality.

We want to improve the scalability by considering clustering techniques to elect deployment positions. Moreover, instead of considering the global trajectories of the drones, we want to look at the problem just between consecutive time steps. This means determining how the drones should move, given their current positions and future sensors' positions. Although we cannot ensure the global optimal solution this way, it becomes a simpler problem which can scale better. In addition to scalability, this approach also provides a higher adaptability. We can consider new information, such as drone failures, at each time step. Consequently, this approach can rely on estimating the sensors' positions in the next step, which is more practical than knowing their initial trajectories. However, the quality of the solution depends on the quality of the estimation and how we deal with bad estimations. Nonetheless, we believe this approach will be an exciting subject to study. We also want to consider more heuristics and different optimisation techniques.

For the second problem, we proposed a model to find the drones' trajectories to recharge fixed sensors in two steps. This division means our solution is not optimal. However, the complexity of the model is much lower. The first step is a mixed-integer linear program to identify which positions the drones must visit and how long they must spend to recharge all sensors. The second step is an heuristic to determine the drones' trajectories to visit the selected positions. We considered that the drones could charge multiple sensors simultaneously, but we avoided sensors receiving power from multiple drones simultaneously. We did so to ensure energy harvesting efficiency so our model can guarantee that each sensor receives at least its energy requirement.

We need to address two main limitations of our model: scalability and lack of parallelism. We can improve the scalability if we consider heuristics for our first step. We have already simplified our mixed-integer linear program, but it is still the bottleneck for our solution time. Clustering algorithms and known heuristics for the set-covering problem look promising to solve our first step much faster.

Moreover, as we avoid energy harvesting from multiple drones simultaneously, we do not take advantage of the parallelism the drones can provide. Although there is a loss of efficiency, we can recharge the sensors faster with multiple sources. Our solution fails to improve as it should when more drones are available. We have already started to address this issue by using idle drones to help recharge sensors simultaneously in the second step, which means that idle drones can reduce the tasks' duration. However, we believe it will be more efficient to model this non-linear relationship in the first step using either heuristics or non-linear optimisation methods. Another research direction is to relax the constraint that a sensor must be fully recharged. Indeed, we may consider a recharging policy in which we recharge the sensors more often but not at full capacity as long as the battery level remains sufficient for operation. This could open the gate to new solutions using fewer drones.

References

Ackerman, E., & Koziol, M. (2019, May). The blood is here: Zipline's medical delivery drones are changing the game in Rwanda. *IEEE Spectrum*, *56*(5), 24–31. doi: 10.1109/MSPEC.2019.8701196

Aiello, G., Hopps, F., Santisi, D., & Venticinque, M. (2020, August). The Employment of Unmanned Aerial Vehicles for Analyzing and Mitigating Disaster Risks in Industrial Sites. *IEEE Transactions on Engineering Management*, *67*(3), 519–530. doi: 10.1109/TEM.2019.2949479

Altinel, D., & Karabulut Kurt, G. (2016, November). Energy Harvesting From Multiple RF Sources in Wireless Fading Channels. *IEEE Transactions on Vehicular Technology*, *65*(11), 8854–8864. doi: 10.1109/TVT.2016.2515664

Arabi, S., Sabir, E., Elbiaze, H., & Sadik, M. (2018, May). Data Gathering and Energy Transfer Dilemma in UAV-Assisted Flying Access Network for IoT. *Sensors*, *18*(5), 1519. doi: 10.3390/s18051519

Balamurugan, G., Valarmathi, J., & Naidu, V. (2016). Survey on uav navigation in gps denied environments. In *2016 international conference on signal processing, communication, power and embedded system (scopes)* (pp. 198–204).

Bekmezci, İ., Sahingoz, O. K., & Temel, Ş. (2013, May). Flying Ad-Hoc Networks (FANETs): A survey. *Ad Hoc Networks*, *11*(3), 1254–1270. doi: 10.1016/j.adhoc.2012.12.004

Bland, R. G., Goldfarb, D., & Todd, M. J. (1981, December). Feature Article—The Ellipsoid Method: A Survey. *Operations Research*, *29*(6), 1039–1091. doi: 10.1287/opre.29.6.1039

Butterworth-Hayes, P. (2019, April). *Drone delivery operations underway in 27 countries.* https://www.unmannedairspace.info/latest-news-and-information/drone-delivery-operations-underway-in-26-countries/.

Caillouet, C., Giroire, F., & Razafindralambo, T. (2019, June). Efficient data collection and tracking with flying drones. *Ad Hoc Networks*, *89*, 35–46.

Caillouet, C., Razafindralambo, T., & Zorbas, D. (2019, April). Optimal placement of drones for fast sensor energy replenishment using wireless power transfer. In *2019 Wireless Days (WD)* (pp. 1–6). doi: 10.1109/WD.2019.8734203

Cohen, M. B., Lee, Y. T., & Song, Z. (2020, October). *Solving Linear Programs in the Current Matrix Multiplication Time* (No. arXiv:1810.07896). arXiv. doi: 10.48550/arXiv.1810.07896

Commission Delegated Regulation (EU) 2019/945 of 12 March 2019 on unmanned aircraft systems and on third-country operators of unmanned aircraft systems. (2019, March). https://eur-lex.europa.eu/eli/reg_del/2019/945/oj/eng.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

CPLEX Optimization Studio is free for students and academics! (n.d.). https://community.ibm.com/community/user/ai-datascience/blogs/xavier-nodet1/2020/07/09/cplex-free-for-students.

De Donno, D., Catarinucci, L., & Tarricone, L. (2014, July). RAMSES: RFID Augmented Module for Smart Environmental Sensing. *IEEE Transactions on Instrumentation and Measurement*, *63*(7), 1701–1708. doi: 10.1109/TIM.2014.2298692

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (Eds.). (2005). *Column Generation*. Boston, MA: Springer US. doi: 10.1007/b135457

Dias da Silva, I., Busnel, Y., & Caillouet, C. (2023, May). Optimisation de plan de vol de drones autonomes pour une recharge rapide de capteurs. In *AlgoTel 2023 - 25èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications.*

Dias da Silva, I., & Caillouet, C. (2022, May). Quand l'immobilisme coûte plus cher que l'échange. In *AlgoTel 2022 - 24èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications.*

Dias Da Silva, I., Busnel, Y., & Caillouet, C. (2022, December). Trajectory Optimization for Fast Sensor Energy Replenishment using UAVs as RF sources. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference* (pp. 2176–2181). doi: 10.1109/GLOBECOM48099.2022.10001604

Dias Da Silva, I., & Caillouet, C. (2020, June). Optimizing the trajectory of drones: Trade-off between distance and energy. In *2nd International Workshop on Internet of Autonomous Unmanned Vehicles (IAUV) in conjunction with IEEE SECON 2020.* Cuomo, Italy. doi: 10.1109/SECONWorkshops50264.2020.9149781

Dias Da Silva, I., Caillouet, C., & Coudert, D. (2021, December). Optimizing FANET deployment for mobile sensor tracking in disaster management scenario. In *2021 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)* (pp. 134–141). doi: 10.1109/ICT-DM52643.2021.9664204

FCC Starts Rulemaking on Licensed Spectrum for Unmanned Aircraft Use. (2023, January). https://www.fcc.gov/document/fcc-starts-rulemaking-licensed-spectrum-unmanned-aircraft-use.

Feng, W., Zhao, N., Ao, S., Tang, J., Zhang, X., Fu, Y., ... Wong, K.-K. (2020, September). Joint 3D Trajectory Design and Time Allocation for UAV-Enabled Wireless Power Transfer Networks. *IEEE Transactions on Vehicular Technology*, *69*(9), 9265–9278. doi: 10.1109/TVT.2020.2972133

Filippone, A. (2006). *Flight Performance of Fixed and Rotary Wing Aircraft*. Elsevier.

Geraci, G., Garcia-Rodriguez, A., Azari, M. M., Lozano, A., Mezzavilla, M., Chatzinotas, S., ... Di Renzo, M. (2022). What will the future of uav cellular communications be? a flight from 5G to 6G. *IEEE communications surveys & tutorials*, *24*(3), 1304–1335.

Giang, T. L., Dang, K. B., Toan Le, Q., Nguyen, V. G., Tong, S. S., & Pham, V.-M. (2020). U-Net Convolutional Networks for Mining Land Cover Classification Based on High-Resolution UAV Imagery. *IEEE Access*, *8*, 186257–186273. doi: 10.1109/ACCESS.2020.3030112

GLPK - GNU Project - Free Software Foundation (FSF). (n.d.). https://www.gnu.org/software/glpk/.

Gurobi Optimizer. (n.d.). https://www.gurobi.com/solutions/gurobi-optimizer/.

IBM Documentation. (2021, November). https://www.ibm.com/docs/en/icos/20.1.0?topic=cplex-introducing.

Jawhar, I., Mohamed, N., Al-Jaroodi, J., Agrawal, D. P., & Zhang, S. (2017, April). Communication and networking of UAV-based systems: Classification and associated architectures. *Journal of Network and Computer Applications*, *84*, 93–108. doi: 10.1016/j.jnca.2017.02.008

Khan, M. A., Ectors, W., Bellemans, T., Janssens, D., & Wets, G. (2017, January). UAV-Based Traffic Analysis: A Universal Guiding Framework Based on Literature Survey. *Transportation Research Procedia*, *22*, 541–550. doi: 10.1016/j.trpro.2017.03.043

Kwak, J., & Sung, Y. (2018). Autonomous uav flight control for gps-based navigation. *IEEE Access*, *6*, 37947-37955. doi: 10.1109/ACCESS.2018.2854712

Lammers, D. T., Williams, J. M., Conner, J. R., Baird, E., Rokayak, O., McClellan, J. M., … Eckert, M. J. (2023). Airborne! UAV delivery of blood products and medical logistics for combat zones. *Transfusion*, *63*(S3), S96-S104. doi: 10.1111/trf.17329

Li, B., Fei, Z., & Zhang, Y. (2019). Uav communications for 5g and beyond: Recent advances and future trends. *IEEE Internet of Things Journal*, *6*(2), 2241-2263. doi: 10.1109/JIOT.2018.2887086

Mahoro Ntwari, D., Gutierrez-Reina, D., Toral Marín, S. L., & Tawfik, H. (2021, January). Time Efficient Unmanned Aircraft Systems Deployment in Disaster Scenarios Using Clustering Methods and a Set Cover Approach. *Electronics*, *10*(4), 422.

Marques, J. P. P. G., Cunha, D. C., Harada, L. M. F., Silva, L. N., & Silva, I. D. (2021, September). A cost-effective trilateration-based radio localization algorithm using machine learning and sequential least-square programming optimization. *Computer Communications*, *177*, 1–9. doi: 10.1016/j.comcom.2021.06.005

Mishra, D., & Natalizio, E. (2020). A survey on cellular-connected UAVs: Design challenges, enabling 5G/B5G innovations, and experimental advancements. *Computer Networks*, *182*, 107451.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., & Sewell, E. C. (2016, February). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, *19*, 79–102. doi: 10.1016/j.disopt.2016.01.005

Nikolic, J., Burri, M., Rehder, J., Leutenegger, S., Huerzeler, C., & Siegwart, R. (2013, March). A UAV system for inspection of industrial facilities. In *2013 IEEE Aerospace Conference* (pp. 1–8). doi: 10.1109/AERO.2013.6496959

Odonkor, P., Ball, Z., & Chowdhury, S. (2019). Distributed operation of collaborating unmanned aerial vehicles for time-sensitive oil spill mapping. *Swarm and Evolutionary Computation*, *46*, 52–68. doi: 10.1016/j.swevo.2019.01.005

Orfanus, D., de Freitas, E. P., & Eliassen, F. (2016, April). Self-Organization as a Supporting Paradigm for Military UAV Relay Networks. *IEEE Communications Letters*, *20*(4), 804–807. doi: 10.1109/LCOMM.2016.2524405

Ota, T., Ahmed, O. S., Minn, S. T., Khai, T. C., Mizoue, N., & Yoshida, S. (2019, February). Estimating selective logging impacts on aboveground biomass in tropical forests using digital aerial photography obtained before and after a logging event from an unmanned aerial vehicle. *Forest Ecology and Management*, *433*, 162–169. doi: 10.1016/j.foreco.2018.10.058

Otto, A., Agatz, N., Campbell, J., Golden, B., & Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, *72*(4), 411–458. doi: 10.1002/net.21818

Pham, H. X., La, H. M., Feil-Seifer, D., & Deans, M. C. (2020). A Distributed Control Framework of Multiple Unmanned Aerial Vehicles for Dynamic Wildfire Tracking. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *50*(4), 1537–1548. doi: 10.1109/TSMC.2018.2815988

Pugliese, L. D. P., Guerriero, F., Zorbas, D., & Razafindralambo, T. (2016, January). Modelling the mobile target covering problem using flying drones. *Optimization Letters*, *10*(5), 29.

Raivi, A. M., Huda, S. M. A., Alam, M. M., & Moh, S. (2023, January). Drone Routing for Drone-Based Delivery Systems: A Review of Trajectory Planning, Charging, and Security. *Sensors*, *23*(3), 1463. doi: 10.3390/s23031463

Rajendran, S. K., & Nagarajan, G. (2022, March). Network Lifetime Enhancement of Wireless Sensor Networks Using EFRP Protocol. *Wireless Personal Communications*, *123*(2), 1769–1787. doi: 10.1007/s11277-021-09212-6

Saif, A., Dimyati, K., Noordin, K. A., G C, D., Shah, N. S. M., Abdullah, Q., & Mohamad, M. (2021). An Efficient Energy Harvesting and Optimal Clustering Technique for Sustainable Postdisaster Emergency Communication Systems. *IEEE Access*, *9*, 78188–78202. doi: 10.1109/ACCESS.2021.3083640

Shahmoradi, J., Talebi, E., Roghanchi, P., & Hassanalian, M. (2020, September). A Comprehensive Review of Applications of Drone Technology in the Mining Industry. *Drones*, *4*(3), 34. doi: 10.3390/drones4030034

Shakhatreh, H., Sawalmeh, A. H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., ... Guizani, M. (2019). Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access*, *7*, 48572–48634. doi: 10.1109/ACCESS.2019.2909530

Singh, K. N., & van Oudheusden, D. L. (1997, March). A branch and bound algorithm for the traveling purchaser problem. *European Journal of Operational Research*, *97*(3), 571–579. doi: 10.1016/S0377-2217(96)00313-X

S. Lasdon, L. (1970). *Optimization Theory for Large Systems (Leon S. Lasdon) - ProQuest*.

Spielman, D. A., & Teng, S.-H. (2004, May). Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, *51*(3), 385–463. doi: 10.1145/990308.990310

Tang, Y., Hu, Y., Cui, J., Liao, F., Lao, M., Lin, F., & Teo, R. S. H. (2019). Vision-aided multi-uav autonomous flocking in gps-denied environment. *IEEE Transactions on Industrial Electronics*, *66*(1), 616-626. doi: 10.1109/TIE.2018.2824766

Tang, Y., Miao, Y., Barnawi, A., Alzahrani, B., Alotaibi, R., & Hwang, K. (2021, July). A joint global and local path planning optimization for UAV task scheduling towards crowd air monitoring. *Computer Networks*, *193*, 107913. doi: 10.1016/j.comnet.2021.107913

Torresan, C., Berton, A., Carotenuto, F., Di Gennaro, S. F., Gioli, B., Matese, A., ... Wallace, L. (2017, May). Forestry applications of UAVs in Europe: A review. *International Journal of Remote Sensing*, *38*(8-10), 2427–2447. doi: 10.1080/01431161.2016.1252477

Win, M. Z., Pinto, P. C., & Shepp, L. A. (2009, February). A Mathematical Theory of Network Interference and Its Applications. *Proceedings of the IEEE*, *97*(2), 205–230. doi: 10.1109/JPROC.2008.2008764

Xu, Y., Liu, Z., Huang, C., & Yuen, C. (2021, December). Robust Resource Allocation Algorithm for Energy-Harvesting-Based D2D Communication Underlaying UAV-Assisted Networks. *IEEE Internet of Things Journal*, *8*(23), 17161–17171. doi: 10.1109/JIOT.2021.3078264

Yang, Z., Xu, W., & Shikh-Bahaei, M. (2020, February). Energy Efficient UAV Communication With Energy Harvesting. *IEEE Transactions on Vehicular Technology*, *69*(2), 1913–1927. doi: 10.1109/TVT.2019.2961993

Yang, Z., Yu, X., Dedman, S., Rosso, M., Zhu, J., Yang, J., ... Wang, J. (2022, September). UAV remote sensing applications in marine monitoring: Knowledge visualization and review. *Science of The Total Environment*, *838*, 155939. doi: 10.1016/j.scitotenv.2022.155939

Zeng, Y., Xu, J., & Zhang, R. (2019, April). Energy Minimization for Wireless Communication With Rotary-Wing UAV. *IEEE Transactions on Wireless Communications*, *18*(4), 2329–2345. doi: 10.1109/TWC.2019.2902559

Zhang, S., Qian, Z., Kong, F., Wu, J., & Lu, S. (2015, April). P3: Joint optimization of charger placement and power allocation for wireless power transfer. In *2015 IEEE Conference on Computer Communications (INFOCOM)* (pp. 2344–2352). doi: 10.1109/INFOCOM.2015.7218622

Zolanvari, M., Jain, R., & Salman, T. (2019, December). Potential Data Link Candidates for Civilian Unmanned Aircraft Systems: A Survey. *IEEE Communications Surveys & Tutorials*, *PP*, 1–1. doi: 10.1109/COMST.2019.2960366

Zorbas, D. (2020, September). *Powercast P1110/P2110B efficiency calculator.* https://github.com/deltazita/powercast-calculator.

Zorbas, D., Di Puglia Pugliese, L., Razafindralambo, T., & Guerriero, F. (2016, November). Optimal drone placement and cost-efficient target coverage. *Journal of Network and Computer Applications*, *75*, 16–31. doi: 10.1016/j.jnca.2016.08.009

Zorbas, D., Raveneau, P., & Ghamri-Doudane, Y. (2016, December). Assessing the Cost of RF-Power Harvesting Nodes in Wireless Sensor Networks. In *2016 IEEE Global Communications Conference (GLOBECOM)* (pp. 1–6). doi: 10.1109/GLOCOM.2016.7841613

List of Figures

List of Tables

List of Theorems

List of Algorithms

Appendix

APPENDIX A

Scheduling Algorithms

A DB-TOF

The algorithm discussed in Section 6.2.2.

Input: Lists of tasks K such that $u_k \neq \emptyset, \forall k \in K$
Output: A list of tasks O with all tasks and their start and finish time, denoted s_k and f_k respectively
$t = 0$
$\text{Task}(u) = \emptyset, \forall u \in U$
$\text{Position}(u) = \text{base\_station}, \forall u \in U$
$i = |K|$
$O = \emptyset$
while $i > 0$ **do**
  Order K by $\text{Flight}(k)$
  for $k \in K$ **do**
    if $C(k) = \emptyset$ **then**
      $s_k = t + \text{Flight}(k)$
      $f_k = s_k + t_k$
      $\text{Position}(u_k) = p_k$
      $\text{Task}(u_k) = k$
      Remove k from K
    end if
  end for
  $t = \min\{f_k, \text{ such that } k = \text{Task}(u), \forall u \in U\}$
  for all $n \in N$ **do**
    $k = \text{Task}(u)$
    if $f_k \leq t$ **then**
      Add k to O
      $\text{Task}(u) = \emptyset$
      $i = i - 1$
    end if
  end for
end while
return O

Algorithm A.1: Drone-Based Time of Flight (DB-TOF)

B DB-WT

The algorithm discussed in Section 6.2.3.

Input: Lists of tasks K such that $u_k \neq \emptyset, \forall k \in K$
Output: A list of tasks O with all tasks and their start and finish time, denoted s_k and f_k respectively
$t = 0$
$\text{Task}(u) = \emptyset, \forall u \in U$
$\text{Position}(u) = \text{base\_station}, \forall u \in U$
$i = |K|$
$O = \emptyset$
while $i > 0$ **do**
  $\text{assign\_tasks} = (|K| > 0)$
  while assign_tasks **do**
    assign_tasks = *FALSE*
    Order K by $\text{WaitTime}(k)$
    $k = \text{first}(k \in K)$
    if $\text{Task}(u_k) = \emptyset$ **then**
      $s_k = t + \text{WaitTime}(k)$
      $f_k = s_k + t_k$
      $\text{Position}(u_k) = p_k$
      $\text{Task}(u_k) = k$
      Remove k from K
      $\text{assign\_tasks} = (|K| > 0)$
    end if
  end while
  $t = \min\{f_k, \text{such that } k = \text{Task}(u), \forall u \in U\}$
  for all $n \in N$ **do**
    $k = \text{Task}(u)$
    if $f_k \leq t$ **then**
      Add k to O
      $\text{Task}(u) = \emptyset$
      $i = i - 1$
    end if
  end for
end while
return O

Algorithm A.2: Drone-Based Wait Time (DB-WT)

C SB-TOF

The algorithm discussed in Section 6.2.5.

Input: Lists of tasks K
Output: A list of tasks O with all tasks and their start and finish time, s_k and f_k respectively, and drone u_k
$\text{Task}(u) = \emptyset, \forall u \in U$
$\text{Position}(u) = \text{base\_station}, \forall u \in U$
$O = \emptyset$
while $|K| > 0$ **do**
  for $k \in K$ **do**
    $u_k = n$ such that n has $\min\{\text{WaitTime}(k)\}$
  end for
  $k = k' \in K$ with $\min\{\text{Flight}(k')\}$
  $s_k = \text{WaitTime}(k)$
  $f_k = s_k + t_k$
  $\text{Position}(u_k) = p_k$
  $\text{Task}(u_k) = k$
  Remove k from K
  Add k to O
end while
return O

Algorithm A.3: Sensor-Based Time of Flight (SB-TOF)

D SB-WT

The algorithm discussed in Section 6.2.6.

Input: Lists of tasks K
Output: A list of tasks O with all tasks and their start and finish time, s_k and f_k respectively, and drone u_k
$\text{Task}(u) = \emptyset, \forall u \in U$
$\text{Position}(u) = \text{base\_station}, \forall u \in U$
$O = \emptyset$
while $|K| > 0$ **do**
  for $k \in K$ **do**
    $u_k = n$ such that n has $\min\{\text{WaitTime}(k)\}$
  end for
  $k = k' \in K$ with $\min\{\text{WaitTime}(k')\}$
  $s_k = \text{WaitTime}(k)$
  $f_k = s_k + t_k$
  $\text{Position}(u_k) = p_k$
  $\text{Task}(u_k) = k$
  Remove k from K
  Add k to O
end while
return O

Algorithm A.4: Sensor-Based Wait Time (SB-WT)

Optimisation du déploiement et de la coordination de drones pour les applications d'exploration et de surveillance

Igor Dias da Silva

Résumé

Les véhicules aériens sans pilote (UAV) ont suscité une attention considérable récemment. Ces appareils volants autonomes présentent plusieurs avantages en termes de coûts de déploiement et de capacités de traitement, ce qui en fait une solution prometteuse pour un large éventail d'applications militaires et commerciales. Dans cette thèse, nous avons étudié deux applications : la création d'un réseau aérien pour connecter des capteurs mobiles à une station de base et la recharge sans fil de capteurs fixes au sol. Dans la première application, nous voulons communiquer avec des capteurs mobiles dans un scénario de catastrophe sans infrastructure de communication préexistante. Notre objectif dans cette application est de déterminer les meilleurs plans de vol pour les UAVs. Les UAVs doivent maintenir un chemin connecté entre la station de base et les capteurs pour permettre aux capteurs d'envoyer des informations critiques si nécessaire. Nous optimisons la distance totale parcourue par les UAVs et l'énergie consommée. La deuxième application concerne un ensemble de capteurs au sol fixes que nous devons recharger. La technologie de récupération d'énergie, basée sur l'échange de signaux radio-fréquence, permettant de recharger les batteries des capteurs sans fil. Nous avons proposé une solution en 2 étapes où d'abord, un programme linéaire détermine où chaque drone doit aller et combien de temps il doit y rester pour s'assurer que les capteurs reçoivent suffisamment d'énergie. Ensuite, des algorithmes gloutons d'ordonnancement déterminent l'ordre dans lequel les UAVs visitent ces points, en veillant à ce qu'ils ne se gênent pas mutuellement ou ne réduisent pas l'efficacité de la charge. Nous minimisons le temps total nécessaire pour recharger tous les capteurs dans les deux étapes.

Mots-clés : UAVs, drones, optimisation, coordination, programmation linéaire, algorithmes

Abstract

Unmanned Aerial Vehicles (UAVs) have gained tremendous attention lately. These autonomous flying devices have several advantages in terms of deployment cost and processing capabilities, making them a promising solution for a wide range of military and commercial applications. In this thesis, we have studied two applications: creating a flying network to connect mobile sensors to a base station and wirelessly charging fixed ground sensors. In the first application, we want to communicate with mobile sensors in a disaster scenario without pre-existing communication infrastructure. Our goal in this application is to determine the optimal flight plans for the UAVs. The UAVs must maintain a connected path between the base station and the sensors to allow the sensors to send critical information when necessary. We optimise the UAVs' total distance travelled and energy consumed. The second application has a set of fixed ground sensors we need to recharge. Equipping the UAVs and sensors with power harvesting technology allows us to recharge these sensors wirelessly. We proposed a 2 step solution where first, a linear program finds where each drone needs to go and how long it should stay there to ensure the sensors get enough power. Then, greedy scheduling algorithms determine the order in which UAVs visit these spots, ensuring they do not get in each other's way or reduce the charging efficiency. We minimise the total time to charge all sensors in both steps.

Keywords: UAVs, drones, optimisation, coordination, linear programming, algorithms