TÉCNICO LISBOA

Task Allocation Algorithms for a Cooperative Drone Parcel Delivery System

Tomás Miguel Bernardo Bastos

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Bruno João Nogueira Guerreiro
Prof. Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Bruno João Nogueira Guerreiro
Member of the Committee: Dr. Meysam Basiri

December 2020

Acknowledgments

I would like, in the first place, to thank Prof. Bruno Guerreiro and Prof. Rita Cunha for the advisory during the time I dedicated to this thesis, and for all the fantastic suggestions and fruitful discussions, without which I would not have accomplished the quality I wanted for this work. I extend this gratitude to all the research team I had the privilege to work with: João, Gil, Parth and Hassan.
A huge acknowledgment to my family, especially to my parents Isabel and Pedro and to my brother Francisco for giving me everyday support and for creating a wonderful environment for me to fulfil my life objectives. Also to Luna, for pushing me to always do better and for all the attentiveness, encouragement, comfort and friendship. To these four people, also a big appreciation for being so inspiring. I would also like to dedicate this thesis to my grandparents Nazária, Deolinda, Manuel and Manuel for always being an example of character and values.
Also a big thanks to all my friends who were part of my life journey until now, without whom I would not be so active, enthusiastic and so happy as I am today. A special thanks to Miguel, for always having my back in all circumstances and to Tomás and Rodrigo with who, besides all the friendship, I had profitable conversations and discussions not only about the subject of this thesis but also regarding other matters and important decisions during all my university path.

This work was partially funded by the FCT project REPLACE (LISBOA-01-0145-FEDER-032107) which includes Lisboa 2020 and PIDDAC funds, and also projects LARSYS (UIDB/EEA/50009/2020) and CTS (UIDB/EEA/00066/2020).

Resumo

Nas últimas décadas o tema de sistemas multi-agente e mais precisamente sistemas multi-*robot* tem vindo a ser amplamente estudado com o aumento não só de aplicações práticas na vida diária de tais sistemas como também pelo aumento constante de possibilidades com o avanço da tecnologia.
Paralelamente, logística e sistemas de distribuição em cidades modernas tem também sofrido um grande investimento e dedicação, dados os problemas das grandes cidades de hoje em dia com enfoque na poluição e sustentabilidade, trânsito ou até nos benefícios económicos de novos sistemas de transporte. Empresas privadas e entidades públicas têm vindo a encontrar nos veículos aéreos não tripulados, também conhecidos como *drones*, uma solução para alguns destes problemas não só em logística urbana mas também em missões de salvamento ou de vigilância.
Esta tese enquadra-se num projeto de investigação no âmbito de sistemas de distribuição através de *drones*, e consiste num estudo de algoritmos de optimização para a alocação de tarefas numa equipa de *drones* para transporte de encomendas. Com este fim, vários algoritmos são estudados, com enfoque no recente algoritmo *Task Sequential Greedy Algorithm*. Este algoritmo foi modificado para inclusão de limitação de bateria dos agentes, possibilidade de recarregamento, com objectivos parametrizáveis pelo utilizador, podendo conjugar tempo e energia gasta. O algoritmo foi também modificado para a implementação de manobras de *relay*, onde, numa manobra em voo, a encomenda pode passar de um drone para outro. Um algoritmo de optimização binária foi implementado para a decisão de onde e quando as manobras de *relay* são mais benéficas para o sistema.
Casos de estudo específicos são apresentados, analisando uma aplicação real do trabalho realizado numa perspectiva do utilizador do algoritmo desenvolvido, bem como uma detalhada explicação da implementação destes algoritmos e das suas modificações.

Palavras-chave: Optimização; Sistemas de transporte por drones; Alocação de tarefas; Manobras de *relay*; Sistemas cooperativos.

Abstract

Multi-agent and multi-robot systems have been being widely studied in the last decades, not only due to the practical applications in real quotidian life, but also due to the constant growth of application possibilities arising with technological advances.

Moreover, delivery systems and urban logistics have also experienced a great investment and effort, given the modern city problems such as excessive pollution, lack of sustainability, increasing road traffic or even due to the financial benefits of modern transportation systems. Both private companies and public entities have found in drones a viable solution not only for urban logistic problems but also for search and rescue and surveillance missions.

This thesis, developed in the context of a research project regarding drone parcel transportation systems, addresses a study of optimization algorithms for task allocation in a fleet of drones with the goal of parcel delivery. With this goal, many algorithms are discussed, focusing on the study and implementation of Task Sequential Greedy Algorithm. This algorithm was modified to include drone battery limitations and recharge possibilities with a configurable objective function giving the user the possibility of combining and tuning both time and energy consumed. It was also modified to include relays, where parcels can change carrier during an in-flight maneuver. A binary optimization algorithm is implemented to decide where and when the relay maneuvers are beneficial to the system.

Specific case studies are presented, analysing a real application of the generated algorithm from a user point of view as well as a meticulous and detailed explanation of all decisions made and of the implementation of the algorithm and its modifications.

Keywords: Optimization; Drone transportation systems; Task allocation; Relay maneuvers; Cooperative systems.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

This work was developed within the scope of the project REPLACE [1], which aims to study new strategies of parcel delivery in urban environments by drones with the possibility of cooperation between agents and relay maneuvers, which includes a variety of technological challenges in research areas such as optimization, planning, estimation, and control. This work focuses on studying and designing algorithms for the planning and scheduling of a set of tasks for a set of Unmanned Air Vehicles (UAVs), part of the planning and optimization challenges within REPLACE project.

The inclusion of relay maneuvers in the cooperative parcel delivery is one of the main and disruptive key steps of the research team during the development of this work, and one of the goals was to include relay maneuvers in the task allocation algorithm (the definition of these maneuvers will be stated in Chapter 5).

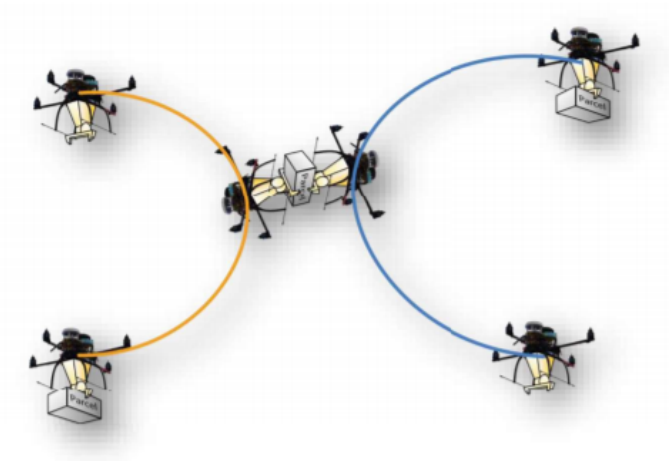

Figure 1.1: Example of Relay maneuver

Numerous applications of Multi-Robot Systems have been emerging in our society and are becoming

more and more generalized, mainly in situations where some places cannot be easily reached or there are human lives in danger, such as helping in preventing and fighting wild fires. Many expect that this type of use will evolve to a common utilization of these types of systems in quotidian tasks rather than only in calamities situations. For example, Amazon Prime Air, the Amazon's delivery system projected precisely with drones or DHL breakthroughs with Parcelcopter (the name of the designed vehicle) having 4 versions today with many months of real environment testing. Also Domino's Pizza has already beta projects to start with drone delivery. More recently the Federal Aviation Administration granted Amazon approval to use drones to deliver packages. Further information and a review of the most recent applications and projects regarding this subject can be found in [2], [3] or [4].
Given this, the motivation to the research effort on the topic of Multi-Drone cooperative parcel delivery strategies and algorithms is the real world application of this research and also the fact that it is solving new challenges regarding modern transportation services and the latest technological challenges in this area.

1.2 Objectives

The proposed work is to design or implement algorithms using concepts of deterministic or global optimization, in such a way that given a number of parcels and a given fleet of drones, all the parcels are transported from the pickup place to the delivery destination in the minimum time possible or with the minimum energy spent. Different algorithms and solutions may be implemented and compared in order to achieve the best approach.
Cooperative task allocation refers to an environment where many agents work together to perform the tasks that are asked to the team, covering all the cooperation operations such as carrying a parcel by more than one drone or planning a path to cover an entire area in a surveillance mission, for instance. Regarding delivery systems, and this thesis in specific, the work produced aims to study, in a first stage an algorithm that retrieves the optimal scheduling of the drones in an environment where heavy packages might be transported by more than one drone, with limited energy and recharge possibility, and in a second stage the implementation of the possibility for the drones to change the parcel carrier in the air (as illustrated in Figure 1.1).
In this context, there are two main objectives of this work:

1. Build a cooperative environment with energy limitation and recharge procedures

2. Implement relay maneuvers on the built environment

To fulfill these objectives, the state-of-the-art literature is reviewed and the associated algorithms are analysed, in order to choose one (or several parts of different approaches) to be modified to include the proposed implementations. It is also an objective to create a good framework for data visualization regarding the proposed problem. After this, another objective is to analyse the performance of the created algorithms (with the modifications) and understand its limitations and real world applications.

It is also worth noting that MRTA (Multi-Robot Task Allocation) problems are divided in two main areas: the decision making of the group of agents and its execution. The presented work focuses only on the first part. This means that the drones are assumed to have the capability of performing the trajectories planned, not colliding with each other. The mechanisms to well perform attitude control and collision avoidance systems are out of the scope of the present thesis. Moreover, the drones are considered to be in a 2 dimensions plane.

There are also some general objectives considering this thesis. First of all, and given that this work is being conducted inside one research project that will continue beyond this thesis, one of the general objectives is to develop the code and the algorithm in such a way that it can be used by anyone and such that independent modules can be added in the code regarding other study subjects, such as a real model of a UAV or a different graphic package or even a module to improve the study to a 3D analysis. Secondly, another important objective is to keep the program and the algorithm generic enough to be able to solve the optimization problem regardless of the type of input data (e.g. homogeneous or heterogeneous fleets).

1.3 Thesis Outline

In order to better follow the line of thought of this work, in this section a brief explanation of each chapter will be done.

Chapter 2 will focus on the analysis and synopses of the more recent advances in literature and international conferences regarding multi-agent systems and more precisely multi-robot task allocation. A part of the section will be dedicated to applications of these algorithms specifically to drone environments. In the same chapter a review of the literature on relay maneuvers and multi-hop transportation systems will be performed.

In Chapter 3, a theory background wrap up on relevant subjects, such as Graph Theory, Optimization and general algorithms, is made. Graph Theory is important for modeling and visualizing task and drone dependencies, as well to understand some constraints of the algorithms. In the topic of general optimization, a generic introduction is made followed by some paragraphs on how to categorize optimization problems. To conclude, specific algorithms, that are relevant to the work developed in this thesis, are explained.

Chapter 4 includes a deep analysis of fundamentals algorithms used on the proposed strategy and a step by step explanation of how it is implemented as well as the reasoning behind the modifications and extra implementations that are done. In the end, the computational performance of the algorithm (run time study) is addressed as well as a case study with a practical application of the developed algorithm from a client point of view.

Regarding Chapter 5, relay maneuvers are implemented, discussing the best strategy to implement these maneuvers, with a step by step explanation of all decisions taken. In the end, a case study is presented with a heterogeneous fleet.

Finally, Chapter 6 addresses the main conclusions of the developed work, as well as some considera-

tions regarding future work that be done based on what was developed in the context of this thesis.

Chapter 2

Topic Overview and State-of-the-Art

As referred, UAV applications and usage possibilities have been greatly increasing in the last decade. As a consequence, studies about Multi-agent Systems (MAS) and more specifically Multi-Robot Task Allocation (MRTA), which include UAVs, are also becoming more and more frequent and precise, both with industrial, commercial or security applications. In general, these problems aim to find a near optimal solution to some cooperative behavior of the group of agents, having in consideration particular constraints that vary from problem to problem. The growing research and study on this topic led to a growing complexity of the algorithms. Thus, the state-of-the-art research has as main objectives the reliability of the systems, its performance, and the simplicity of the algorithms.

2.1 Cooperative Task Allocation

MRTA problems can be sub categorized: the first division we can make is whether if each robot is performing just one task or multiple task in parallel (number of tasks per robot simultaneously). Some work regarding single task robots have used Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP), such as [5] and [6]. In the first case, air vehicles are expected to identify, classify and target ground objects. Because of that, the tasks need to follow a specific order, having strict time and precedence constraints. Most of the work that we will analyze are single task robot studies, since with this setup we can address most of the objectives of this work. Other articles dedicated to this topic propose an approach based on Particle Swarm Optimization (PSO), as we can see in [7], [8], [9] and [10]. The authors of [7] combine PSO and a Genetic Algorithm for a weapon target task allocation for UAVs in battlefield environment. The work produced in [10] analyses scheduling in an indoor 3D situation in order to minimize the total energy consumed. The data set of tasks in this paper (and others) is defined by its starting and end position, due time interval, and predecessors, with an environment that includes recharge stations. Other algorithms (Restful Task Assignment Algorithm, for example) are used to find one feasible solution before the optimization, and just then PSO is used, and in the end, various data sets with different characteristics are tested and the results analysed. A similar study is conducted in [9].

We can also divide the problems and algorithms regarding the number of robots in each task: single robot tasks or, in opposition, tasks that require more than one agent to be performed (cooperative missions). Even though the majority of the approaches we see in the literature are single robot task, we will focus mainly in the second type on the development of our work. One article that analyses a cooperative task allocation using also PSO is [8], which provides the best sequence of tasks to minimize the cost function, defined as the mission completion time. A Sequential Greedy Algorithm is presented as a non optimized solution to practical task allocation problems. After, PSO is applied to obtain a better solution and both solutions compared. Sequential Greedy Algorithm is used in [11] as a baseline comparison, and some changes to that algorithm are proposed to reach better solutions, generating a Task based Sequential Greedy Algorithm that will be very important in the development of this thesis. The last two papers will be crucial on the initial approach of the solution proposed in this thesis. On top of that, more work is being dedicated to cooperative task allocation as we can observe in [12], where a multi-structure Genetic Algorithm is implemented in a fleet with different types of UAVs.

Additionally, some work have been conducted on task allocation under uncertainty environments as we can observe in [13] and [14]. In [13] we can analyse both a stochastic programming and robust optimization strategy, used generally when data needed to the stochastic model is scarce, with the main objective of minimizing the worst cost scenario. The two approaches are then studied together, generating an analysis on its relationship and benefits. This is done by defining each tasks by two different parts: one deterministic and one where uncertainty is created, depending on the workload of the task. Also in [14] a Filter Embedded Task Assignment algorithm, doing reassignment each time the surrounding conditions are updated, is presented and also a mixed formulation of this algorithm with a robust optimization approach, with a tuning parameter through which the user can modify the magnitude of the uncertainty. Numerical large scale simulations are also conducted using Monte Carlo Simulation methods.

Another important way to sub divide multi-agent problems is the organizational paradigm. The first approach, a centralized individual decides the final solution for all the time span of the mission and for all agents, with the information regarding each agent (either calculated or communicated by the robots). Alternatively, there are decentralized processes where agents communicate with each other their information on the environment, taking a group decision of which tasks are going to be performed by each of the agents. This approach is generally connected to a market-based approach to the task allocation algorithm where agents bid for each task, based on economic market theory. Many stages and designs can be implemented using decentralized approaches, with the main advantages of time and computational resources efficiency, operation under unknown and uncertain environments and robustness with respect to model uncertainty. On the other hand, centralized algorithms often demand high computational resources for big missions, and so they commonly have a problem regarding scalability. Also, in what concern robustness of the process, it all relies in one central agent an so, if some problem occurs with that one, all system fails. In an example of using a specific type of ILP, [6] describes the implementation of Binary Linear Programming and iterative Network Flows and Auction algorithms for the single assignment problem and analyse its scalability to multiple assignment problems. In [15] and

[16] a new decentralized and cooperative approach of the task allocation problems is presented, using POMDP (a better explanation can be found in the next paragraph), showing that this process can also be applied to decentralized problems, specially when not too many agents are being concerned. To grow the population of agents, great independence of agents needs to be assumed. Furthermore, a distinct decentralized approach is presented in [17], where an auction algorithm is extensively explain and compared with other algorithms: Consensus Based Bundle Algorithm (CBBA).

Another algorithm used in this types of allocation problems is Markov Decision Processes (MDP) and Partially Observable Markov Decision Processes (POMDP) when the agents can only partially observe the simulation environment. Thus, in POMDP, uncertainty is always considered. In [18], a standart MDP is used to a task allocation problem. It is also shown the scalability of this process to greater number of agents and tasks. [19] combines a MDP with reinforcement learning where the algorithm tries to discover the optimal policy for the task allocation, which is better than just an heuristic optimisation, according to the conclusions of the work.

In what concerns modern urban logistics and city delivery systems, which is the main propose of application of the work developed in this thesis, many other articles have been presented to the scientific community inspired by optimizing economy costs in parcel delivery, tackling the problem of excessive traffic in the modern cities or reducing the gas emissions and transportation sustainability impact. Examples of these are [20] or [21] where taxi fleets are used to transport not only people but also delivery parcels. Also public transportation was studied with the objective of taking advantage of the cargo space that is not used in these vehicles such as the work presented in [22] and [23].

Despite the numerous publications and studies regarding multi-agent systems, transportation strategies and more precisely drone cooperative task allocation for parcel delivery, there are few that incorporate energy limitation and recharge possibilities and for this reason it is believed that this thesis represents a great advance in the study of these subjects.

2.2 Relay Maneuvers

With the increasing interest in the subject of city logistics, modern delivery processes studies have developed in the last 2 decades, and more recently, disruptive concepts have been proposed. In this context, there has been several authors investigating systems were many agents contribute to perform one single delivery task, with a change in the carrier during the task performance itself. These approaches are called multi-hop parcel delivery systems and are not very-well studied despite having great advantages and potential if one can create an efficient system.

In [24], Chen *et al* present an approach with a complex ILP formulation solved by two different heuristics, with the goal of using the spare capacity of existing transportation flows, incorporating the option of transfers between drivers. In this article is also demonstrated that these systems' performance increase with the number of vehicles and parcels participation. Similarly, in [25], Hong *et al* study an incentive based approach to a Multi-Hop parcel delivery network, based on historical traffic data, and so the aim of the study is that anyone is able to carry parcels and drop them somewhere between the pick up

point and its destination, until the parcel reaches the delivery destination. The problem is tackled by two different points of view: the first one from the platform point of view where rewards to the carriers are optimized in order to minimize costs, and from the user point of view with the goal of maximizing its profit.

This subject have also been studied in other fields of science such as machine learning or artificial intelligence, as in [26] where it is presented a model based on reinforcement learning through artificial neural networks to optimize scheduling of routes and admission of parcels to the system.

Despite the fact that research in this subject is still rapidly growing, it is important to understand that these approaches are based on existing vehicle routes (although sometimes the drivers can have an incentive to change their route in order to fulfill a mission). This means that a parcel and the associated delivery task is assigned to an available vehicle, and its schedule or route for the vehicle is, in the majority of the cases, not changeable.

In parallel, relays in multi-robot systems are also under intense study in the context of Industry 4.0, where smart transportation is one of the main focuses (see [27] or [28]). Relay maneuvers are geographic approximations of agents (mainly drones in our literature review) that are used as a tool to help in the plan and decision upon the agents schedule and trajectory. Its applications in literature focus mainly on cooperative agents regarding communication network planning, surveillance or video covering of live events and the research regarding this subject is growing as well as its utility and relevance [29]. In communicative systems, the planning of distance between agents is crucial, once the antennas have a limited transmission and reception radius. The objective of the path planning algorithms is to overlap the areas of communication of the UAVs in order to ensure a network communication that successfully transmits information between the agents. Examples of this are the works performed by Kopeikin, Ponda *et al* in [30] and in [31]. In the first example the authors propose the application of a distributed algorithm named Consensus-Based Bundle Algorithm with relays to perform the path planning and task allocation whereas in the second example the authors use the same algorithm (CBBA with relays) but include realistic data entries such as path loss or stochastic fading. Wang also presents an example in [32] of a path planing of drones to cover sports live events, with relay approximations to maximize coverage. Another example of relay planning, this time for military missions using drones, is presented in [33].

Concluding, multi-hop transportation systems are in focus in the literature in the last years, but the studies are being conducted with the goal of using free space in existing vehicle routes and not regarding task allocation and path planning of a cooperative drone systems. In fact, when talking about drones and relay maneuvers, the studies focus more on telecommunication and network execution whereas literature about parcel delivery drone systems with relay procedures, i.e a systems where drones are able to change carrier in flight, are extremely scarce. For this reason, the work presented in this thesis is believed to include a great innovation regarding studies in multi-robot systems as well as drone parcel delivery systems.

A summary of the most relevant approaches studied in this section is shown in Appendix A, with the main goal of aiding the reader in the research of different algorithms.

Chapter 3

Background

3.1 Graph Theory

3.1.1 Introduction

There are lots of problems that can be easily represented and modeled by points and connections between them in order to ease its visualization and analysis. These are useful as they can illustrate almost every relationship or structures between objects or agents. For example representing houses in a city and communication cables between them, or the numerous airports in Europe and the scheduled flights of an airplane. These diagrams are called graphs and are mathematical entities that have been the focus of a deep analysis and study in discrete mathematics.

Considering a more formal definition, as defined in [34], a graph $G = (V, E)$ is composed by a set of vertices V (also called nodes) and a set of edges E, where each edge has two nodes associated. In the example, Figure 3.1, nodes are represented by the letters u, v, w and x while edges by the letters from a to f. Endpoints are defined to be the nodes that delimit each edge, e.g u and v are endpoints of a. Two edges are adjacent if they have an endpoint in common. Edges can have directions depending on the problem formulation (for examples in a road, the direction in which the traffic flows). If this is the case, it is called a digraph and the endpoints might be called head and tail, depending on the direction of the edge so that the direction points from the head to the tail.

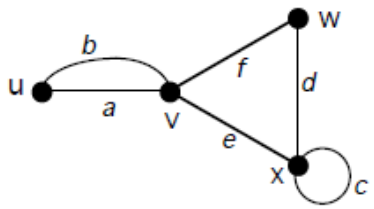

Figure 3.1: Graph example [34]

Continuing with some useful definitions, with the analysis of a cooperative transportation problem on mind: we say that a graph or sub-graph is a walk in the form of the sequence $\{v_0, e_1, v_2, e_2, ..., v_n\}$ if for

every $n = 1, ..., n$, v_{n-1} and v_n are endpoints of e_n. In other words, a graph is a continuous sequence of adjacent edges. A walk is said to be closed if the the initial node coincides with the last node (i.e $v_0 = v_n$). Also, we call trail to a walk that has no repeated edges ($e_i \neq e_j$ with $e, j = 1, ..., n$ and $e \neq j$). Similarly, a path is a trail that does not repeat any internal node (initial and final nodes can be the same and if this is the case, we say it is a closed path).
Taking the previous image as example, we can say that:

- $\{u, a, v, b, u, a, v, f, w\}$ is a walk but is not an trail, neither a path;
- $\{u, a, v, f, w, d, x, c, x\}$ is a walk, is a trail, but is not a path;
- $\{u, a, v, b, u\}$ is a walk (closed), is a trail and is also a path.

Finally we say that a closed path is a cycle, as the last walk in the previous enumeration.

3.1.2 Directed Acyclic Graphs

Directed acyclic graphs will have a great relevance in the algorithm definition for the problem of cooperative parcel delivery.
First of all, remembering the definition of cycle, we say that a directed acyclic graph (also DAG) is a digraph that has no cycles (no closed paths). As a consequence, we can say that in order to be a DAG, every walk inside a graph must be a path. A result of this is that in every DAG, it is possible to attribute numbers to the vertices from 1 to n in such a way that for any selected edge directed from v_i to v_j, it implies that $j > i$. This process is called topological sorting. These graphs arise from natural formulations with temporal or hierarchy ordering in fields like sociology, causal structures, biology, computer software or operations research as it will be presented in the following paragraphs.
Let's take the example of sociology: an organizational chart in which we take employers as nodes and hierarchy relationships as edges, is a DAG, as no lower level employee has a relation where he is superior than an upper level employee. Also in genealogy, a family tree (or a pedigree chart, often used to study genetics) is a DAG if we model the direction of the edge from the older individual to the younger as seen in Figure 3.2. The organizational chart example can be also seen as this image upside down.

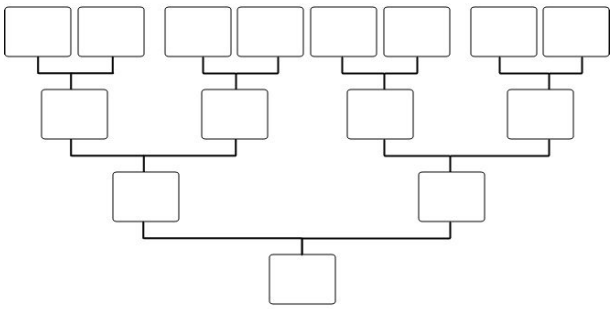

Figure 3.2: Family Tree

Imagine also a software design task, where functions A,B and C are defined. It is impossible to formulate function A recursively with function B, and function B with relation to function C if function C is recur-

sively defined with function A. We would get a cycle inside the software that could not be resolved. A similar situation appears in the problem proposed to be studied in this work: cooperative task allocation algorithms.

Let's assume we have 2 drones a_1, a_2 and 3 tasks t_1, t_2, t_3 to be performed. Task 1 only needs 1 agent to be performed whilst in tasks 2 and 3, both drones are needed to complete the tasks (because of the weight of the parcel for instance). Let's say that the vector V_n stores the output of the ordered tasks to be performed by the agent n after some task allocation decision was made: $V_1 = (1, 2, 3)$ and $V_2 = (3, 2)$, meaning that drone 1 performs task 1, then task 2 and in the end task 3 (represented in red), and that drone 2 performs task 3 and then task 2 (represented in green).

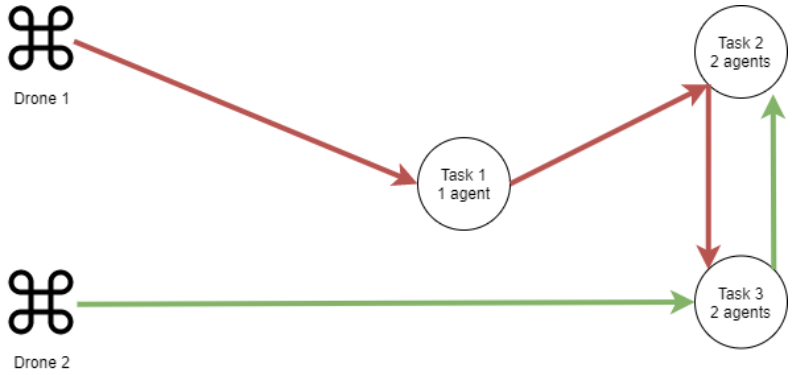

Figure 3.3: Example of task allocation representation

We can directly analyze the dependency of the tasks: from drone 1 path, task 3 depends on task 2 to be executed, and task 2 depends on task 1. On the other hand, given drone 2 task allocation, task 2 is dependent on task 3. We can easily observe that task 2 and task 3 have a mutual dependency and thus, it is an impossible solution for our problem. In the following image we can observe the time dependency of the tasks on this example, knowing that an edge directed from t_n to t_m means that task m need task n finished to begin (time hierarchy).

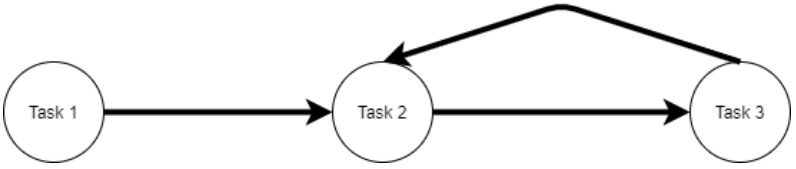

Figure 3.4: Task dependency graph

This graph has directed cycles and thus it is not a DAG, which is the reason why it cannot be a solution of a cooperative task allocation problem, has we have seen before. In Chapter 4, further discussion and analysis will be held, having this conclusion in mind.

A viable solution to this problem would be for instance $V_1 = (1, 2, 3)$ and $V_2 = (2, 3)$, as we would not have a mutual task time dependency anymore. Drone 1 would perform task 1 and then, drone 1 and 2 would cooperatively perform task 2 and 3.

3.2 Optimization

Optimization is generally described as a procedure of making the best or most effective choice given an objective and a set of limited resources. It has been present in nature long before human studies on operations research began. Physical systems tend to optimize (minimize) its energy: take springs systems or electrons states of energy as an example or even bird groups triangular formation that reduce global drag.

Along with the development of society, culture, and science, optimization problems started to arise as populations wanted to achieve the best profit (whatever that meant in each situation) with their scarce resources. More recently, in modern mathematics, particularly in the area of operational research or decision science, optimization problems have been a focus of a myriad of studies and applications in all science and industry fields. These efforts emerge as companies want to optimize procedures to reduce costs, investors want to optimize their portfolios to maximize returns, mechanical engineers want to maximize product characteristics, minimizing the weight and materials used.

Before the study of each optimization problem, 3 quantities should be identified: objectives, variables and constraints. This is labeled as the modelling procedure. After this, a systemic procedure should be set to solve the specific model. This procedure is called algorithm and we will analyse some examples in this chapter.

It is also essential to formally express the optimization problem. Following, one can observe a generic mathematical formulation, according to [35]. Note that this formulation will be followed in the present thesis, and that this example is given in the minimization form, without prejudice to the generality of the problems.

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subjected to} \quad & \mathbf{x} \in C \end{aligned} \tag{3.1}$$

being C the set of constraints, that are generally a set of inequalities extensively described in literature as $g(\mathbf{x})$ functions. Note that $\mathbf{x} = (x_1, x_2, x_3, ..., x_n)$, the variable vector, can have as many variables as the user desires depending on the problem modelling. The objective function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is the quantity the user wants to optimize and depends on the optimization variables. Also, in this case we present a single objective function, whereas there are some problems that are set to be a multi-objective optimization problem.

It is usually useful to classify optimization problems, in order to identify and study the best approach to a desired problem. Some of the most applied categories will be described next.

One way to categorize an optimization problems is regarding the nature of the optimization variables, whether it is continuous or discrete (i.e if $\mathbf{x} \in \mathbb{N}$ or $\mathbf{x} \in \mathbb{R}$). One classical example of discrete optimization is a factory producing 2 types of products, that has to decide how many of each to produce considering profit and resources of each one. On the other hand, for example a mathematical optimization of a continuous variable is the minimization of energy (electricity) used in a production line. A special and relevant type of discrete optimization is integer optimization and a special type of this is binary

optimization.

Another widely used differentiation of optimization problems is between linear and non-linear optimization. Mathematically, a problem is said to be linear if the objective function and all the constraint functions are linear. Note that a function $f(x)$ is linear if it satisfies the condition:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{3.2}$$

Also, it is common to classify these problems as constrained or unconstrained optimization, regarding having explicit constraints on the variables or not. Is is also common and sometimes advantageous to transform constrained problems into unconstrained by the means of penalty coefficients in the objective function. The constraints on the variables may vary from simple intervals to complex relationships between them.

Another two relevant classifications are single or multi-objective optimization and deterministic or stochastic optimization. The majority of study problems of optimization are single objective, being computed a score function with different weights in the variables, but multi-objective optimization techniques are rising in modern research. These types of problems appear in areas such as engineering or in economics and reach different solutions to the models, each one representing a trade off between the optimization variables. For example in performance engineering (motorsport for example), decision makers want to be able to clearly see different options with different approaches and choose between all of them, performing a trade off strategic decision. The most common visualization of these different solutions are called Pareto Fronts or Pareto Curves and represent the different solutions in the solution space. For example let us imagine a financial analyst that wants to perform an optimization on a stock portfolio. He wants to decide where to invest based on 3 different portfolio characteristics: risk index (minimize), expected variation (maximize) and maximum possible payoff (maximize). Instead of creating a function that outputs just one ”score” computing these characteristics, there are algorithms that retrieve many points: the point of minimum risk for each expected variation and maximum payoff, the point of maximum expected variation for each risk index and maximum payoff and the point of maximum maximum possible payoff for each risk index and expected variation, creating a surface with all of these points and clearly giving the user the option of choosing which point to choose, knowing the trade offs that it implies.

Considering deterministic versus stochastic optimization, the differentiation relies on the data and information modelling. In deterministic optimization, data is considered to be accurately known, meaning that it will not change or vary during the optimization process and is strictly connected to reality. In stochastic optimization, data is considered uncertain. This can happen for numerous reasons for example physical systems with sensors errors or models that rely on data prediction. There are many algorithms of optimization under uncertainty, depending on the degree of confidence the user has on data correctness (maximum degree of confidence tends to deterministic optimization). This separation might also be defined regarding the nature of the implemented algorithm.

In order to solve optimization problems, specific types of approaches are studied for each type of prob-

lems. It is important to analyse the problem itself, but also to choose an adequate procedure. In this context, an algorithm is described to be a sequence of steps that receives an input (or a vector of inputs) and returns an output after some processing procedure. From an optimization point of view, an algorithm is a systematic and repetitive procedure generated to process a set of information and solve an optimization problem, retrieving a solution. There are algorithms created to solve a specific problem, others created to solve a group of problems (e.g linear programming, transportation and assignment problems) or even general algorithms applicable to any problem such as meta-heuristics. Further discussion on this topic will be held in Section 3.2.3. In the following sections, some relevant and generic algorithms will be described and analysed.

It is important to understand that algorithm creators and users seek for algorithms that are not only efficient but also correct. The efficiency of an algorithm describes the amount of resources, namely computational effort, that is needed to perform the algorithm. It is important to choose or create an efficient algorithm not only because computational power is a limited and expensive resource in real problems and applications, but also because of their run time, that lowers when algorithm efficiency is increased. Concerning correctness of the problem, it quantifies how close to the optimal solution the algorithm can get. It is extremely important in algorithm creation and implementation to justify why the author or the user believes that the algorithm solves the problem correctly. This task is often difficult because in most cases the optimal solution is not known. Sometimes mathematical proof is possible, however sometimes it is only possible to support the algorithm correctness based on logic reasoning or extensive verification and validation.

Another extremely important characteristic of an algorithm is its scalability, i.e the capacity of the algorithm to maintain its performance regarding efficiency when the number of input variables or the size of the data set is increased. This is an important characterization in the algorithm design phase in order for the user to better understand the limitations of the procedure and also in the developer point of view in order to try to reduce the algorithm complexity. One of the most used notations to classify and compare the algorithms in terms of scalability is the Big O Notation, for which the theoretical background can be found in [36] and [37]. The necessary comprehension regarding this subject, having in mind its practical application to our work and analysis, is the order notation and what it represents. For example, saying that an algorithm is $0(n)$ (it is read "Big O of n") means that the complexity (normally the run time) of the algorithm varies in the same order of magnitude of n, where n is usually the dimension of the optimization variable or some other variable used to define the cost function or the constraints. This means that when we increase the size of the optimization variables or input data, the complexity increases linearly. On other hand, saying that an algorithm is $0(n^2)$ means that the complexity increases with n squared, having a quadratic evolution with the increase of the size of the inputs. In the following table it is possible to observe some of the most common Big O Notation and the characterization of its evolution. The entries are ordered from the lower increase in complexity with the increase of n, to the higher dependency.

Big O	
Notation	**Evolution**
$O(1)$	Constant
$O(log(n))$	Logarithmic
$O(n)$	Linear
$O(nlog(n))$	Loglinear
$O(n^2)$	Quadratic
$O(n^c)$	Polynomial of order c
$O(c^n)$	Exponential
$O(n!)$	Factorial

Table 3.1: Common Big O Notations

3.2.1 Greedy Algorithm

Some fundamental algorithms will be presented and discussed in the scope of the work presented. The first one is Greedy Algorithm, described by Cormen, Leiserson, Rivest and Stein in [38] as ”algorithm for optimization problems [...] that always makes the choice that looks best at the moment”. This means that at each decision step, the algorithm judges all the options and opts for the one that, in that step is leading to a better solution. It is easy to show that a greedy algorithm does not always reach an optimal solution, and in some cases it does not reach even a good one, as seen in the next example.

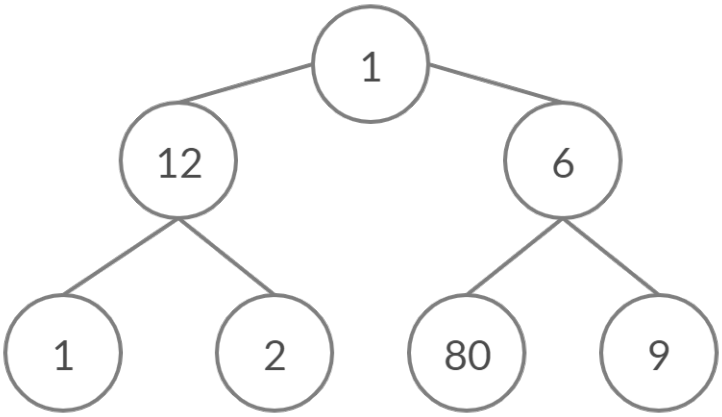

Figure 3.5: Tree graph example

Let us consider that the objective function is to maximize the total score (the sum of each step’s score). Considering the image 3.5. Greedy algorithm would, in the first decision making, chose the place that scores 12 and in the second decision making would chose the place with score 2. We can easily observe that the optimal solution would be 87 points, choosing 6 in the first step and 80 points in the second. The algorithm will reach 15 points.

There are although some circumstances and models where a greedy algorithm encounters an optimal solution. For example the problem of finding the minimum coins to give change in euro currency. It can easily be solved by fitting the coin with highest value at each step. As an example, consider that we want to provide €1.23 as change. We try at each step to find the highest coin below the total value. In this example, first we choose €1, then with the 23 cents left we choose 20 cents, and after 2 and 1 cent,

giving the optimal solution. Another examples of problem where greedy algorithm reaches an optimal solution is the fractional knapsack problem (further information in [38]), or the minimum spanning tree algorithm explained in Chapter 9 of [39], with numerous applications such as network cables planning.

3.2.2 Transportation and Assignment Algorithms

In literature regarding operations research there is broad study around transportation and assignment algorithms. In this section, those algorithms will be explored as it is extremely important to understand the reasons why they do not apply to our optimization problem and goal. Extensive information that goes beyond what will presented next can be found in Chapter 8 of [39]. Both the referred algorithms are specific applications of linear programming, with specific characteristics. However, simple linear algorithm as Simplex would require an enormous computational time once these problems are also large and with numerous variables and constraints.

Transportation problems are considered to be a problem where goods or parcels should be transferred from one location (supplier) to the another (destination). The goal is to transport the goods in a optimal way, regarding not only which routes to use but also how many of each good to transport in each one of them. The classical example of these problems is when a company has n factories producing a supply and m warehouses demanding a certain amount of that supply. Each travel between one of the n factories to one of the m warehouses has a modeled cost and the goal is to optimally decide how much supplies comes from each one of the factories to each one of the warehouses. It has relevant applications not only in transportation but also in production planning or employers schedule as an example.

Assignment algorithms are a special type of transportation problems, with the aim to assign tasks to people, which at first sight is exactly the work proposed in this thesis, if we assume that to the knowledge of the algorithm, the assignees can be either people, but also, machines or vehicles. However, some relevant restrictions and particularities appear in these formulated algorithms.

Transportation problem model assumes for example that each supplier has a fixed number of supply units and that all the supplies must be assigned and the same happens with the destinations, having a fixed number of demand units, and all have to be satisfied. It is of course very limitative to imagine a problem where a set of suppliers produce the exact quantity of product that the destination would need, and for this reason, there are strategies to go around the fact of all units must be satisfied both in supply and destination such as the called ”Dummy destinations and supplies” where virtual entities are created and assigned or the ”big M modelling method” were big penalties are charged for each of the requests that is not met. But the fixed number of supplies and demands remains, which is a condition that in our problem does not apply, because different drones might have different carrying capacity which implies that the number of drones that a parcel need, to be carried varies. The model also assumes that the cost of a transportation is proportional to the number of units transported, whereas we can easily imagine a real model of a drone where the energy and power needed do not vary linearly with the quantity that is being transported or even a situation where drones can decide whether to travel faster of slower

depending on the number of tasks they have ahead and upon the conscious of being or not the limitative agent in a specific situation. meaning that they would be able to decide to travel slower to save energy.

In what concerns assignment problem model, some assumptions are also considered. The ones that are not possible to overcome, in the context of our problem, are that each task is supposed to be performed by one and only one agent and that each agent should perform one and only one task. These assumptions go strictly and directly against the definition of a cooperative system, against the fact that there will possibly exist more tasks than agents and against a time continuity of the algorithm.

Also, the algorithms are being studied to be used in dynamic environments, within a framework that is supposed to allow for the implementation of sequential levels of complexity such as battery limitation of the agents, recharge locations, relay maneuvers, tasks appearing continuously in time and also the inclusion of a real drone model and study of uncertainty in this context, which does not allow for the use of classical assignment and transportation algorithms and even deterministic approaches at a certain level.

3.2.3 Metaheuristic Optimization: Genetic Algorithm

In the matter of optimization problems, there are some characteristics of it that might hamper the application of specific algorithms. For instance, non-differenciable problems, high dimension problems or non-linear problems may not be fit to use algorithms such as gradient search methods, combinatory algorithms or linear algorithms (such as Simplex Method or Transportation and Assignment algorithms) - see [39] for more information on these algorithms.

For this reason, the scientific and technical communities felt the need to use heuristic algorithms. The word *heuristic* comes from the ancient Greek language and was used to refer to new ways and strategies to solve problems. In this context, an heuristic is an iterative procedure that, beginning in a feasible solution of the problem, tries to increase the optimality of the solution in each sequential iteration. The global optimality of the process depends on the computational capability of the machine that is solving the problem. Heuristics are usually created to a specific problem and the advances in the field shown that this approach is very effective in some cases. However, as they are dependent on the characteristics of the problem itself, they usually are not fit to be used in other applications, meaning that with the appearance of a new problem, one must create a new algorithm (or adapt an existing one).

On this basis, metaheuristics were created in the late 1980's. The word *meta* means "upper level", and in this context refers to strategies and algorithms that are not defined to a specific problem. Instead, it is a general procedure used to solve many types and families of problems. They do, however, share some characteristics with heuristics as they are also iterative and do not assure acquisition of the optimal solution of the problem in study.

Metaheuristics can be divided in two main groups: single solution based or population based (a specific case of those are Evolutionary Algorithms). In the first case, a single solution is iterated and compared with the previous one, searching the space of solutions with this single solution. We can denote Tabu Search or Simulated Annealing as examples of these. Alternatively, in population based methods,

there is a group of solutions called population, that is changed and analysed as a whole throughout the iterations of the algorithm, until a certain stoppage rule is met. This can be either set by quality of the encountered solution, percentage of the solution space searched or just limit number of iterations achieved. Examples of this group of metaheuristics are Particle Swarm Optimization or Genetic Algorithms. All of these algorithms can be revisited in [39]. Among these, we will further discuss Genetic Algorithm (GA) as it will be useful in the work developed in the context of this thesis.

Evolutionary algorithms, and GA in particular are founded on natural and biological principles such as the theory of evolution proposed by Charles Darwin [40]. The next paragraphs will be used to explain and discuss some natural *phenomena* worth visiting for the understanding of GA.

Darwin's theory is widely acknowledged in literature and scientific community, and has its basis on a principle called Natural Selection. This selection is performed by nature itself in the species and populations because, as Darwin states, the most fit individuals to a specific environment will have more chance of surviving (a *phenomenon* called Survival of the Fittest). This leds to more probability of generating offsprings, i.e reproducing, and the characteristic that concedes the comparative advantage in that specific environment is, thus, perpetuated and spread to more and more individuals in the successive generations. This conclusion assigns to the species a capability of adapting to changing environments and naturally choosing the best individual attributes as a response to this change.

It is worth noting that Darwin did not have the information we have today on the matter of genetics. Some of the discoveries that came in the following century validated this theory and explained some natural processes that Darwin did not explain, such as the biological processes for inheritance of parents characteristics, the way that genetic information is stored in our cells or even the appearance of mutations in the reproduction process (important for enhance the adaptability of the species to new environments).

Genetic theory applied to computational algorithms and procedures was firstly proposed by Holland in [41] in 1975. Following, the basic GA is going to be scrutinised as well as the procedures and processes in it. Let us first define some useful terms in the context of evolutionary algorithms:

1. Individual/Chromosome: In this context individual or chromosome refers to the unit that composes the population. Each individual or chromosome is composed by a set of parameters that together encode a solution to the problem in study. In biology, a chromosome is a piece of DNA that encodes genetic information of the individual. Individuals from the same species have the same amount of chromosomes. There are also some versions of Genetic Algorithms where each individual is composed by more than one chromosome, but this work will not go into that detail [42].

2. Population: Population is a group of individuals that constitutes the evolutionary identity of the algorithm. It is inside this population (group of individuals) that crossover will occur in other to give birth to the second generation, and also inside which will occur the selection of the fittest, given the fitness function of the problem.

3. Gene: Inside each individual, there is a finite number of genes and each one is connected to a specific parameter of the problem.

4. Fitness: Fitness refers to the value of the fitness function (defined for each problem) of one individual. It correlates to the biological ability of surviving and reproducing in one environment.

5. Natural Selection: Biologically is the natural mechanism that presses the most fit characteristics of the individuals to perpetuate throughout generations. In computation it is represented in the selection process.

More subdivisions of genes can be executed, for more complex problems, for examples alleles. However, for the application in this context there is no need to go further on the divisions.
In the Figure 3.6 it is possible to observe a representation of the above mentioned terms. This example has also another characteristic which is the genes only possess binary values.

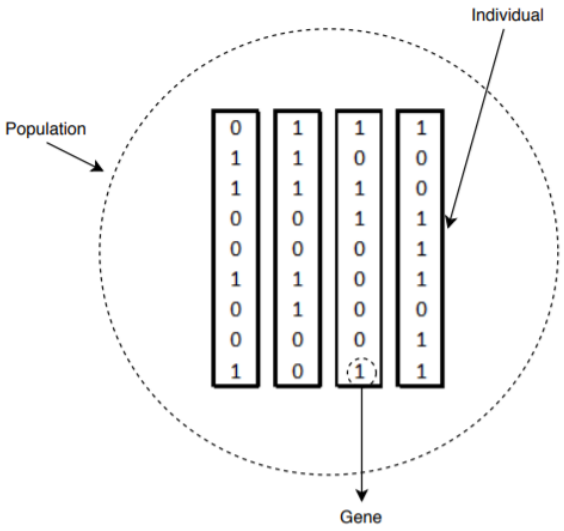

Figure 3.6: Genetic Algorithm definitions illustration

As referred, GA is a population-based, stochastic method that is iterated from step to step and ends whenever a termination criteria is met. It consists in 3 main phases: selection, crossover and mutation.
The selection process contains the *phenomenon* of survival of the fittest, and it is the main ”motor” of genetic algorithms. In this process, the strongest (according to the fitness function) individuals will be chosen (selected) to reproduce and create the next generation that will be a combination of their own genes (perpetuation of the best characteristics). It is also worth noting that this is also a stochastic process in order to maintain diversity of the results and to run from local maximums/minimums. In fact, the most fit individuals are generally only attributed a higher probability of reproduction and not deterministically chosen although there are several selection strategies.
Roulette wheel selection is the simplest one and attributes probabilities of being selected proportional to the fitness score of the individual. We can imagine as an example that some numbers from 1 to 100 are attributed to each individual, the more fittest the more attributed numbers to that individual. After this, random numbers will be selected to be the next generation parents. Other strategy that ensures more diversity is Ranking Selection, where individuals are ranked according to their fitness value and after that,

individuals from all ranks are chosen, in a linear proportion, this is, more individuals from higher ranks and less from the lower. Also Elitism strategies are used quite often when convergence of the algorithm is to be accelerated, often coded as a percentage of the total population. With this approach, after ranking the individuals the algorithm will choose the fittest ones to directly pass to the next generation without having crossover and mutation processes. This obviously enhances the convergence but prejudicing the diversity of the solutions.

Regarding Crossover phase, it is an operator that forms the offspring's chromosome starting from the information stored in its parents. Regularly, 2 parents generate 2 new offsprings in order to maintain population dimension; both offsprings are most probably different from each other and different from its parents. This process is the responsible for the search of new solutions in the solution space. In biological reproduction the genetic information crossover occurs in a process called meiosis, and the way that information is replicated is complex. In the computation algorithm, there are many methods to do the genetic coding. The most simple are One and Two point crossover. Analysing the chromosome as a vector, one point crossover selects one point of the vector, splitting it in two sections, and attributes the first parent's section to one offspring and the second section to the other, and the other parent the other way around. The Two point crossover acts in a similar way, but divides the parent chromosome in 3 different sectors, as Figure 3.7.

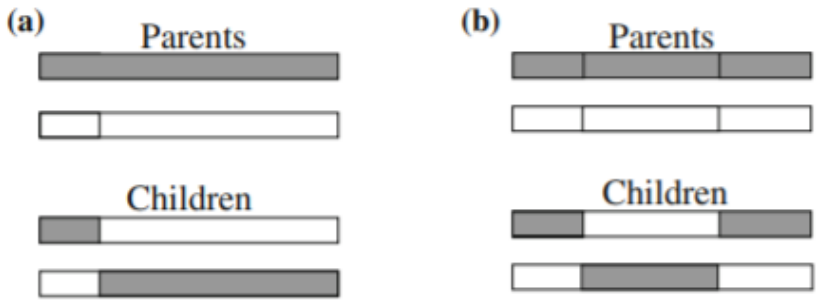

Figure 3.7: Crossover methods illustration: a) One Point; b) Two Point [43]

It is also possible multi-point crossover by splitting the chromosome in more parts. Uniform crossover refers to a process that, after splitting the parents' chromosomes, the passing section to each children's is sorted randomly.

The last procedure that we will describe, mutation, occurs to fight the non capability of the crossover process of generating new information or also diverging from a premature convergence. It can occur both in the crossover process by changing the information of one gene, or by a creation of a offspring from one only parent, with a change of one of its genes. It can be done by substituting, inserting or deleting a gene. They can vary both in the new introduced genetic information as in the position of the mutation in the chromosome. Generally mutations are computed to have a very low probability of occurring, but its occurrence guides the algorithm to overcome local maximums, the so called hill-climbing capability, thus helping the search of the global solution space.

Regarding the stochastic nature of the meta-heuristic, all probabilities must be tuned by the user in order

to maximize the capacity of search of the algorithm, preserving the convergence but also the diversity of the solutions, within the available computational power. This parameters are for instance the probability of mutation, the type of selection, the type of crossover or the percentage of the population that will become parents of the next generation. These parameter should also be an input to the algorithm, that can be observed in Algorithm 1.

Algorithm 1 Genetic Algorithm

In: *Algorithm Parameters, Objective Function*
Out: *Best individual* ▷ Best solution for the problem

1: **procedure** GENETIC ALGORITHM
2: Population initialization (P_0) ▷ Any feasible solution
3: $t = 0$ ▷ Generation counting
4: **while** *Stop Criteria* not met **do**
5: Evaluate individuals fitness
6: Selection
7: Crossover
8: Mutation ▷ generation $T = t + 1$ created
9: Active population actualization
10: $t = t + 1$
11: **return** Solution ▷ Only returns the best individual

Chapter 4

Task Allocation for Parcel Delivery with Recharging

After careful review of the state-of-the-art and selected literature, it was considered that a good first step would be to study in depth the algorithms presented in [11] by Oh, Kim, Ahn and Choi, given that the present work is the first step into the study of the inclusion of recharge procedures and relay maneuvers in an algorithm for cooperative task allocation. Others of the studied approaches would also be good and viable, however, this algorithm was chosen given that this work aims to provide a first study about this topic, that can be after modified to tackle scalability, robustness or other issues that might appear after a solid framework is implemented. The first section of this chapter will be an overview of the work described on this paper, followed by the implementation of the proposed algorithms with a number of changes on one hand in the formulation and on the other hand adding some new features and capabilities to these algorithms such as recharging capability. This change enhances the possibility of analysing real problems and study the behaviour of the team of agents in a realistic simulation environment and with real logistic constraints and objectives. Finally some simulation results will be discussed with previously generated data sets.
To display algorithms and pseudo-code, whenever necessary, the pseudo-code convention proposed in [44] will be used, complying with its format and notation.

4.1 Basic Algorithm Overview

This paper, similarly to this thesis, is a response to the increasing UAV mission demand, given the problem that sometimes the a parcel delivery task might imply that the task weight is greater than a drone capability. For this reason, this article comes up with a solution for a single task multi-robot problem, many times referred as a coalition formation problem in the literature. We will define coalition as the group of drones working together to perform a task such that each task will have a associated agent coalition in the end of the algorithm.
First of all, it is pertinent to formulate the problem regarding this work, that can be generally extrapolated

to the major problem formulation of this thesis. The mathematical framework such as variable nomenclature might be altered from the original article in order to guarantee coherence along the entire thesis writing. Despite this, in this section the original framework and problem statement will be presented, and the particular modifications for this thesis work will be introduced in the following sections.

Let us imagine a 2 dimension environment with N agents (drones) and M delivery tasks. Each drone $i \in \{1, 2, 3..., N\}$ is initially characterized by its initial x and y coordinates x_i and y_i respectively, its payload maximum cargo weight, c_i, operational times for pickup and drop-off t_i^p and t_i^d and average velocity v_i. Regarding task $k \in \{1, 2, 3, ..., M\}$ initialization, the user has to provide information about its pickup and drop-off locations (x_k^p,y_k^p,x_k^d and y_k^d), weight of the cargo w_k and time window for delivery $[t_k^i, t_k^f]$. In the next figure we can observe the idealized environment in the analyzed paper. Next to each task location we are able to observe the number of agents needed to complete it and a p or d, specifying if it is a pickup or drop-off point. It is worth, at this points, clarifying some nomenclature along the thesis: the term "user" will be used to describe the person that is expected to use the developed algorithm to perform the optimizations (the client of our work) and the word "programmer" will be used to refer to the person that is developing the program itself.

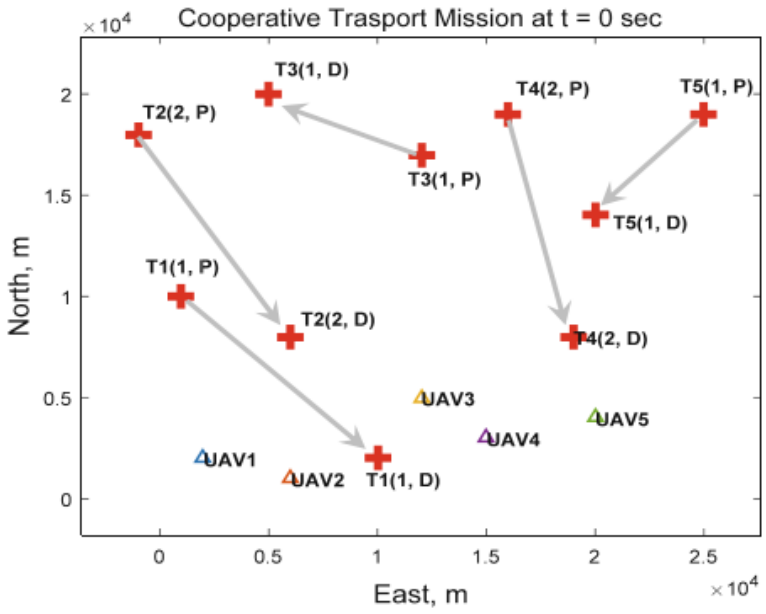

Figure 4.1: Mission environment [11]

Within this framework, **P** is defined as the optimization variable that stores the various tasks each agent will perform, in order. We can say, by this definition, that **P** is a list or vector of vectors that stores the duty of each agent, in the order in which the tasks will be performed. The dimension of **P** depends on the number of agents and on the number of agents needed to perform each task. For instance, given an environment with 4 agents, with 10 tasks that need all agents to be performed, **P** will be a matrix 4×10. Also, to each coalition (group of agents that will perform task k) the author calls $\mathbf{a_k}$. For example, $\mathbf{a_2}$ refers to the group of agents that are set to perform task number 2. Thus, saying $\mathbf{P}_{2,3} = 4$ means that the task number 4 will be the 3^{rd} one performed by agent 2. Generally, $\mathbf{P}_{i,m} = k$ means that task k

will be the m^{th} one performed by agent i. On the other hand saying that $\mathbf{a_k} = (m, n, o)$ means that the agents m,n and o will perform task k.

The score function associated with a single task is defined as

$$s_k = \begin{cases} s_0 + \frac{t_k^f - t_k}{t_k^f - t_k^i}, & \text{if } t_k^i \leq t_k(\mathbf{P}) \leq t_k^f \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Equation (4.1) means that if a task is performed between the time intervals, the score will be a sum of a default score value s_0 and a variable score that is bigger when the task is completed sooner in the admissible time. On the other hand, if the task is not completed in this time interval or not completed at all, the score is 0. We can calculate $t_k(\mathbf{P})$ for task k as follows:

$$t_k(\mathbf{P}) = \left(\max_{i \in \mathbf{a}_k} t_{ETA}(i, k) \right) + t_p + d_k/V + t_d \tag{4.2}$$

where $\max_{i \in \mathbf{a}_k} t_{ETA}(i, k)$ is the time of the last agent arriving at the task which corresponds to the task initial time. Remember that it is necessary to have all the agents in a_k in the pick up point to initiate the task. Also t_p and t_d are the operational times for pickup and drop-off, d_k is the total distance of the task and V is the average speed of the drones.

Given this, the presented problem can be represented as follows:

$$\begin{aligned} \text{maximize} \quad & J = \sum_{k=1}^{M} s_k \left(t_k(\mathbf{P}), t_k^i, t_k^f \right) \\ \text{subjected to} \quad & \sum_{i \in \mathbf{a}_k(\mathbf{P})} c_i \geq w_k, \quad \forall k \in \{1, 2, 3..., M\} \\ & \text{is}\mathrm{DAG}(G) = 1 \end{aligned} \tag{4.3}$$

It is worth analyzing the score function $s_k \left(t_k(\mathbf{P}), t_k^i, t_k^f \right)$ that for each task computes the score associated with it, having as inputs the time when the task will be completed, $t_k(\mathbf{P})$, and the admissible time interval for completion of the task t_k^i and t_k^f. Besides this, *G* is the graph generated by the dependencies between the tasks as analysed in Section 3.1.2.

Analysing the constraints of this formulation, the first one states that the sum of the payload capacities of the coalition formation for a specific task must at least meet the weight of the parcel. The second constraint ensures that *G*, the graph generated by the task allocation result, is a directed acyclic graph as we have defined in Section 3.1. The conceptual function $\text{is}\mathrm{DAG}()$ is simply a function that returns 1 if the input graph is a directed acyclic graph and 0 if not.

The main focus of Oh *et al* in [11] is the analysis of an algorithm already implemented in [8] and the suggestion of some changes in it, in order to study if it improves the results considering the problem addressed in this work. The main differences in the problem formulation presented in [8] rely on the fact that there is no score function and the objective function is simply to minimize the total mission time (i.e. the time corresponding to the last completed task). Another minor difference is the formulation of the maximum payload constraint: instead of having a weight per parcel and assuring that the total

capacity of the coalition, each task as an additional parameter expressing the number of agents needed to perform that task. Is assumes an homogeneous fleet, whereas equation (4.3) is a general formulation that can comprise heterogeneous fleets. The formulation of [8] can be observed in Section 4.3, as it will be the starting point of our study. Given these differences, the two different algorithms proposed by Oh *et al* will be presented next.

4.2 ASGA vs TSGA

The first algorithm is presented in [8] and called Sequential Greedy Algorithm (SGA) for cooperative timing missions. The coalitions in this algorithms are chosen in a greedy procedure, among all possible coalitions. For this, the algorithm computes the ETA (Estimated Time of Arrival) of all pairs *(agent,task)*, and chooses the best of these times to decide the next task to be allocated and the coalition leader for those tasks, which are the pair *(agent,task)* with minimum ETA. After this, until the sufficient number of agents, the agent with minimum ETA to that task is allocated sequentially. This procedure is repeated until all tasks have an allocated coalition. This algorithm is purely greedy and, as we have seen in Section 3.2, might be far from the optimum. Despite this, an important characteristic of this procedure given the cooperative nature of this problem, is that it automatically ensures that the DAG constraint is met, because each task is allocated consecutively in the end of each agent schedule, preventing the appearance of crossed dependencies.

In [11], modifications to this algorithm are proposed. In order to differentiate both algorithms we will adopt the nomenclature proposed in the paper. Therefore, the next algorithm (from [8]) will be called Agent Sequential Greedy Algorithm (ASGA), and after, with the modifications proposed, we will formulate the Task Sequential Greedy Algorithm (TSGA). Following, in Algorithm 2, a pseudo-code presentation of ASGA.

Algorithm 2 Agent Sequential Greedy Algorithm

In: *Tasks, Agents*
Out: *Task Allocation* ▷ Task allocation result

```
1:  procedure SGA(Agents, Tasks)
2:      while unallocated_tasks = True do
3:          Find next task and coalition leader          ▷ Best ETA pairing agents/task
4:          Allocate task to coalition leader
5:          while unsufficient_agents=True do
6:              Find next agent                          ▷ Best ETA agent to the same task
7:              Allocate task to next agent
8:          Actualize agents info                        ▷ Time and position control
9:          Actualize unallocated_tasks
10:     return TA
```

It is worth noting that in ASGA, two major greedy decisions are made. Those are:

- The next task in each step, and thus the order in which the allocation is performed;
- The agent allocation within each task;

The author suggests that the first decision is more important than the second one to the overall performance of the algorithm, because it is the one that establishes the order in which each task will be performed. For this reason, the author suggests changing the order in which the tasks are being considered inside the algorithm, performing an allocation in all possible combinations, storing at each step the best result so far. This means that the first greedy decision disappears. As a result, the capacity to reach a more optimal solution is greatly increased in the algorithm but as a price for that, computational effort is also remarkably increased (actually it grows with $M!$, i.e. M factorial, once we are conducting analysis on all task order permutations). This is a substantial downside of this modification, because for large missions with a great set of tasks, it becomes unfeasible to get a quick result on the allocation procedure. However, as it is a centralized approach, on the one hand the algorithm would not need to be run several times and the user might run it with a big earliness and on the other hand, a vast computational capacity can be installed, without losing mobility of the agents and suitability to the problem we are studying. In algorithm 3 we can observe the main differences between TSGA and ASGA.

Algorithm 3 Task Sequential Greedy Algorithm

In: *Tasks, Agents*
Out: *Task Allocation*

```
1:  procedure TSGA(Agents, Tasks)
2:      Define all possible tasks permutations
3:      while order_of _permutations do          ▷ Test all possible task order combinations
4:          while unallocated_tasks = True do
5:              while unsuficient_agents=True do
6:                  Find next agent
7:                  Allocate task to next agent
8:                  Actualize agents info
9:                  Actualize unallocated_tasks
10:         if mission time < best do                ▷ Control of the best permutation
11:             Actualize best
12:             Actualize TA
13:     return TA                                    ▷ Only returns the best allocation
```

As stated, the second greedy decision inside each allocation is maintained and this keeps the automatic satisfaction of the DAG constraint. In essence, this algorithm runs $M!$ ASGAs with a specific and constrained allocation order. For this reason, there is no need to demonstrate that this algorithm improves ASGA results, because the original ASGA allocation is for sure a subset of TSGA result, noting that the greedy order chosen for ASGA is for sure one of the $M!$ permutations. This ensures that TSGA is in the worst case as good as ASGA. Nevertheless, some numerical simulation results are shown in [11].
To summarize and better compare both algorithms, the fluxograms of both are presented in figures 4.2 and 4.3.

4.3 Computational Implementation and Discussion

In this section the implementation of the algorithms will be explained as well as some decisions and assumptions made during the development of the software. Phyton language was chosen for the implementation of these algorithms and their result analysis. Also, one of the main goals of the algorithms

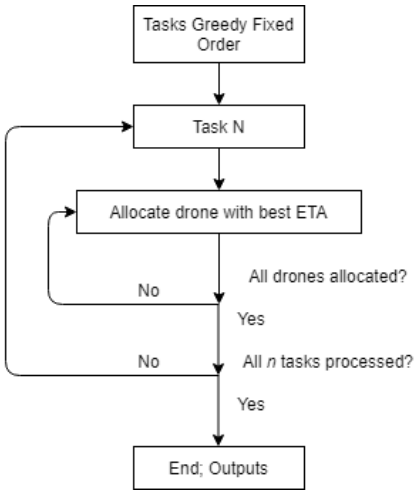

Figure 4.2: ASGA fluxogram

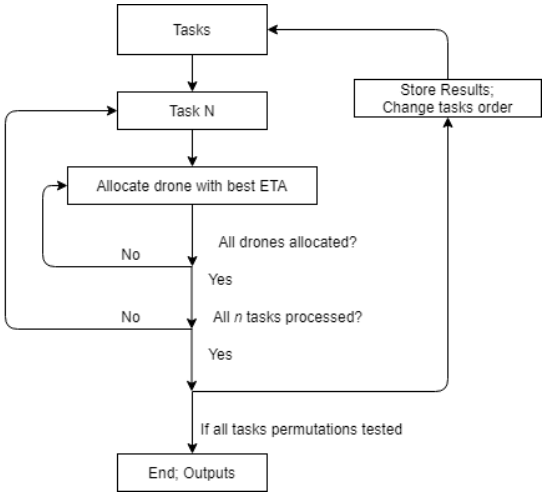

Figure 4.3: TSGA fluxogram

implementation was keeping the process generic enough so that for any type of task set and drone fleet that enters the algorithm, the algorithm will get a good output. For this reason, code development was done step by step with small complexity increases, that will also be explained in this section.

In the first place, some simplifications to the optimization problem were performed, such as having a number of drones associated with each task instead of maximum drone payloads and parcel weight, such as assumed in [8]. For the first approach, operational times for pick-up and drop-off were also ignored as well as the time windows of the tasks. These relaxations do not imply a loss of quality of the results. Accordingly, one formulation of the first approach can be, similarly to [8]:

$$\begin{aligned} \text{minimize} \quad & J = t(\mathbf{P}) \\ \text{subjected to} \quad & n\left(\mathbf{a}_k(\mathbf{P})\right) = Z_k, \forall k \in \mathcal{K} \\ & \text{isDAG}(G) = 1 \end{aligned} \tag{4.4}$$

where $t(\mathbf{P})$ is the total mission time, corresponding to the latest parcel delivery time and Z_k is the number of agents required to perform task k.

With this in mind, the initial agents and tasks information needed to run the program are as defined in Tables 4.1 and 4.2.

Drone initial information	
Info	**Description**
x_pos	x coordinate initial position
y_pos	y coordinate initial position
vel	drone velocity

Table 4.1: Drone class information

Task initial information	
Info	**Description**
x_p	x pick up coordinate
y_p	y pick up coordinate
x_d	x drop-off coordinate
y_d	y drop-off coordinate
na	number of agents needed for task completion

Table 4.2: Task class information

For the implementation procedure and with the objective of following all the modifications and some details of the algorithms and of the results, a data set was created. Analysing each step with a known and constant data set (Data Set 1) will enable a better understanding of the phenomena and account for unexpected behavior. The graphical environment related to this data set is presented in Figure 4.4, where the red circles are the drones initial position, and the crosses are the initial (P) and final (D) points of each parcel delivery task. Regarding the number of agents needed in each task, task 1 and 4 need only one agent to be performed, task 0, 2 and 5 need two agents while three agents are necessary to perform task 3. So that the data set is fully defined it is also worth noting that the velocities of the drones are set to be 0.02 distance units per time units.

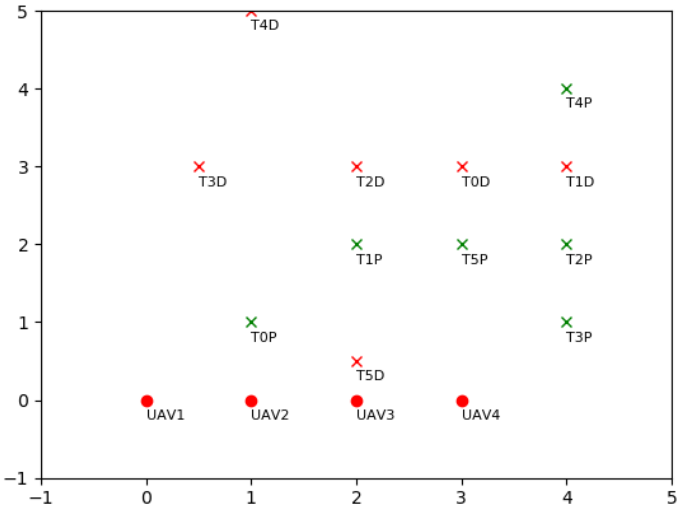

Figure 4.4: Data Set 1 environment

It is worth noting that this data set was almost randomly generated (with some bias to show some

interesting results along the implementation demonstration) and that velocity and distances are not dimensional, and do not have a physical correlation. As noted before, the algorithm is generic enough to input different physical simulation data given that we are considering a generic distance unit and time unit. This means that the user just has to ensure coherence between the used physical units, for example using always meters or always kilometers for the distance units and always second or always minutes for the time units.

Regarding this data set, and after ASGA implementation, the output of the task allocation algorithm is given in table 4.3:

Task Allocation Result - ASGA

Drone	TA
1	[0,3,5,4]
2	[0,5]
3	[3,1,2]
4	[3,2]

Mission time: 1109.93

Table 4.3: Task Allocation 1

This notation for the task allocation result will be used along all this work, and can be interpreted as each row corresponding to each one of the agents (identified in the left column), with the result of the task allocation in the right column. The task allocation list represents the tasks performed by each drone in order. Meaning that, as an example, agent 1 in this problem will perform task 0, then task 3, then 5 and finishes with task 4. For the formulation of this problem, it is important to clarify that each task only begins when all agents needed arrive to the parcel pick up location. This means that all the others that might be available at an earlier time are waiting for that agent to arrive. This waiting time can either be in the previous task dropping point, in the next task pickup point or any point in the middle. In real implementation of this algorithm, and if there is enough waiting time, drones can even pause and rest in order to save battery.

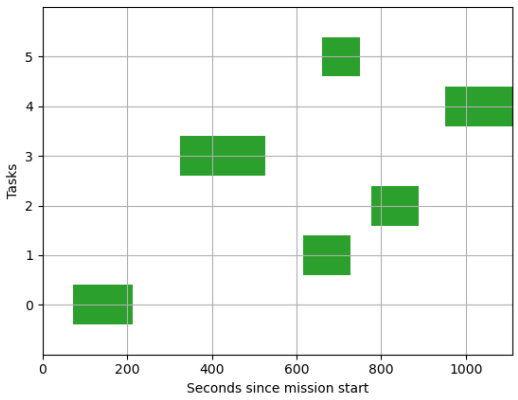

Figure 4.5: Task Execution Times - ASGA

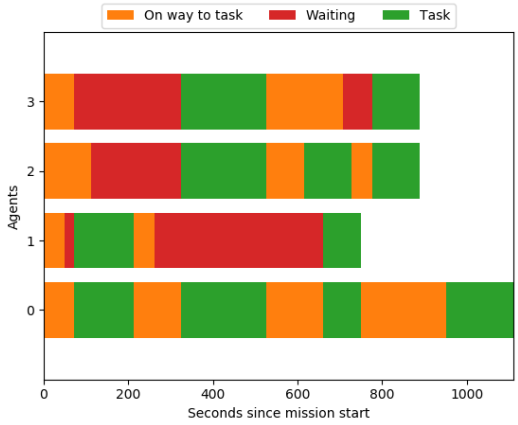

Figure 4.6: Agent Scheduling - ASGA

Graphic features like Figures 4.5 and 4.6 will be very useful during the analysis of the results to understand the algorithm output. The left figure presents the time interval in which each task is being

executed, and the right figure shows the scheduling of each one of the agents, differentiating if they are on way to task, waiting for other drones, or performing a task itself. Note that for these representation assumes that one drone waits (if necessary) for the other coalition drones in the pick up point of the next task, as we may see that it is ”on way to task” before it is ”waiting”. It is also worth noting that the task execution in figure 4.6 (green intervals) coincide with the task execution time at figure 4.5, which might be useful to better understand the scheduling, and also validates the graphic coherence.

After the implementation of Agent Sequential Greedy Algorithm, and with the results retrieved and being a good comparison baseline from now on, Task Sequential Greedy Algorithm was implemented. As stated before, this algorithm tests all the allocation order and outputs the best result. As a result, no changes in input information are needed.

After the implementation of this algorithm, the task allocation result is represented in the Table 4.4. Also, in order to test if the algorithm is correctly implemented, the allocation order for the ASGA was retrieved from the program - (0,3,5,1,2,4) - and after, the result of this specific order in TSGA was also retrieved, to confirm that the task allocation result and mission time corresponds to the ASGA algorithm. In Figures 4.7 and 4.8 we can look at the diagrams for the result of TSGA.

Task Allocation Result - TSGA

Agent	TA
1	[0,5,3]
2	[0,5,3]
3	[2,1,4]
4	[2,3]

Mission time: 656.905

Table 4.4: Task Allocation 2

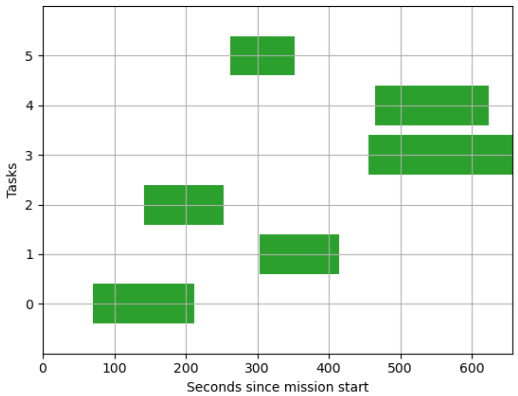

Figure 4.7: Task Execution Times - TSGA

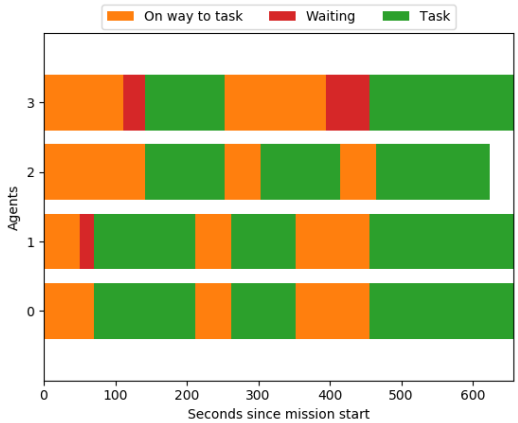

Figure 4.8: Agent Scheduling - TSGA

We are able to clearly observe not only the reduction in mission total time but also a significantly reduction in waiting time of the agents, which was expected because this is nearly the optimal solution for this problem (apart from the greedy decision on the agents allocation to each task). This might be in fact the optimal solution but we do not have a guarantee. Testing all possible agent order in all possible task order would not only be a ”brute force ” search that would need bigger computational resources in

bigger missions, but also, and foremost, this search would not guarantee that the DAG constraint would be satisfied.

4.4 Task Allocation with Recharge Stations

After the above results, the next step in the development of a suitable algorithm for the objectives of this dissertation is the inclusion of a battery limitation on the drones. This implementation increases the degree of reality of the studied environment given that there are no real agents with no energy limitation and also enables the study of more complex missions, where the drone team does not have enough energy to complete all tasks. Reminding the objective of keeping a generic algorithm, capable of running for any drone and task sets, even for a heterogeneous drone (drone fleet with drones with different characteristics), it was added to the class definition of the drones two parameters: maximum energy and energy consumption. These parameters will not have physical meaning, as time and distance scale, and so it was chosen arbitrarily that for this data set, each drone will consume 1 energy unit per distance unit travelled and a maximum energy of 15 energy units per drone.
Concerning how energy consumption is implemented in the code, an ”energy left” parameter was created internally associated to each drone (and restored in each permutation analysis). This parameter is updated every time a task is allocated to the drone schedule. Also, before the allocation of any task, to drone is checked to determine if it has sufficient energy to both travel to the pick up point of the task and to perform the delivery itself. If not, the drone is considered unsuitable for that task as it will not be able to complete the entire task. For this reason, some modifications to the problem formulation might be needed as described in the following paragraphs. It is also worth noting that the mission is considered concluded if all the tasks are performed or alternatively, when all drones are considered not assignable to any of the remaining tasks, meaning that they do not have enough energy to perform any of them.
After implementing this feature, and with (4.4) formulation, the result obtained is as represented in Table 4.5.

Task Allocation Result

Agent	TA
1	[0,4]
2	[0,2]
3	[5,1]
4	[5,2]

Mission time: 440.957

Table 4.5: Task Allocation 3

Note that task 3 is not completed. This happens because some of the allocation orders tested in the Task Sequential Greedy Algorithm do not allow every task to be completed, but in the considered formulation and objective function, we are only minimizing mission time. For this reason, given all the task allocations correspondent to all the possible orders (with and without all tasks completed), the algorithm chooses the minimum mission time that logically corresponds to a mission where less tasks are performed. A quick analysis to every task allocation retrieved during the algorithm process, confirms that other allocation

orders enable the completion of all the tasks. Given this particularity, triggered by the inclusion of the battery limitation, a modification to the actual formulation is proposed, because the main objective of a task allocation procedure is to complete all tasks:

$$\begin{aligned} \text{maximize} \quad & J = f(\mathbf{P}) \\ \text{subjected to} \quad & n\left(\mathbf{a}_k(\mathbf{P})\right) = Z_k, \forall k \in \mathcal{K} \\ & \text{isDAG}(G) = 1 \end{aligned} \tag{4.5}$$

where $f(\mathbf{P})$ is a score function that comprises not only a mission time evaluation but also the number of tasks completed. To enable a deeper analysis and broaden the capabilities and interest of utilization of this software, total energy consumed was also incorporated in this score function. Given this, the implemented score function is:

$$f(\mathbf{P}) = k_1 \times s_{task}(\mathbf{P}) - k_2 \times s_{time}(\mathbf{P}) - k_3 \times s_{energy}(\mathbf{P}) \tag{4.6}$$

Given this score function, the algorithm is maximizing the number of tasks completed, having the mission time and energy as penalties to the score. Note that k_1, k_2 and k_3 are the weight of the task completion, mission time and energy consumption, respectively, and that can be tuned to the preferences of the user. We can continue studying a pure TSGA by setting $k_1, k_3 = 0$, for instance.

The scores $s_{task}(\mathbf{P})$, $s_{time}(\mathbf{P})$ and $s_{energy}(\mathbf{P})$, can also be calculated in several ways, since they are just a measure of performance of the output. In the present work, these scores are calculated as follows.

$$s_{task}(\mathbf{P}) = \frac{number\,of\,tasks\,in\,\mathbf{P}}{number\,of\,total\,tasks}$$

meaning that it is a percentage of completed tasks given the complete task set.

$$s_{time}(\mathbf{P}) = \frac{mission\,time}{mission\,time\,comparative}$$

where $mission\,time\,comparative$ in our implementation is given by the time that a drone with the average speed of the entire fleet would take to perform all tasks.

$$s_{energy}(\mathbf{P}) = \frac{mission\,energy}{mission\,energy\,comparative}$$

where $mission\,energy\,comparative$ is, analogously to $s_{task}(\mathbf{P})$, given by the energy that a drone with the average consumption of the entire fleet would take to perform all tasks.

The last consideration worth noting about this modification is that the value magnitude of each of the scores is not equal, meaning that the absolute value of each contribution may be different. This means that equal coefficients k_i to every score do not imply the exactly same weight of that parcel in the score function. From now on we will set $k_1 = 100, k_2 = 1, k_3 = 1$, that ensures that the task completeness is the major goal of the allocation process, but within a set of allocation results with the same amount of tasks completed, the algorithm will choose the ones with minor mission time and battery consumption.

In order to be sure that the algorithms and the battery modifications were well implemented and to avoid incurring in systematic errors in the analysis due to bad data processing, extensive verification and validation to this and others data sets was done and the results are coherent and comply with the ones obtained from the software implementation. Also, to verify the implementation, the basic task set of one big task (bigger that 15 units) was created and the expected result of a null task allocation result was obtained since there is no drone capable of travelling more than 15 distance units.
The task allocation result is given, with the new formulation of Equations (4.5), represented in Table 4.6:

Task Allocation Result

Agent	TA
1	[0,5,3]
2	[0,5,3]
3	[2,1,4]
4	[2,3]

Mission time: 656.905

Table 4.6: Task Allocation 4

With this modification, the algorithm was able to achieve exactly the same result as before, showing that the implementation was successful. It is also worth noting that this score function is easily changeable. For example, the algorithm does not carry out any preference procedure among the tasks, meaning that if the battery limitation does not allow all tasks to be performed, the tasks that are chosen to be uncompleted are directly related with the correspondent mission time. Despite this, it is simple to include in this score function procedure a ”urgency parameter” to each task. If this algorithm is used to a allocation procedure where there as tasks that represent an higher profit to the mission, that can be easily implemented in the score function and the algorithm will also opt for the most profitable allocation. Similarly, a time interval approach as seen in [11] can also be implemented in the score function.
Following the line of thought of trying to reach simulation conditions as close as possible to the real ones, in order to enable the usage of this code in more complex and interesting missions, the next step is to create the possibilities for the drone to recharge their batteries.
This feature will be implemented with some initial considerations:

- The agents recharge in specific places on the map called bays;
- The recharge is instantaneous (one can think of a battery swap, as a parallel to a real process) and do not have the operational time of approaching and leaving the bay; this means that the drone recharges just by equalling its coordinates to the coordinates of a charging bay;
- The bays do not have a limited space for drones to recharge, meaning that any bay is always available for any drone to recharge and there are no queues or waiting on the recharge process
- When a recharge happens, the drone battery becomes full;
- An agent is not capable of recharging while performing a task.

It is clear that some of this assumptions are a relaxation of the reality, meaning that in a real world

environment they would not occur, but they keep the sufficient reality to preserve the relevance and fidelity of the study of the task allocation procedure.

Having this considerations in mind, regarding the coding inside the algorithm, the recharge can be seen as a task with the initial point being the drop off point of the previous task and the final point the coordinates of the bay. The first decision made regarding this implementation was that the recharge task would be allocated to the drone schedule in the middle of the task allocation procedure. Another option would be retrieve a task allocation matrix from the TSGA and after, in a post processing stage, a procedure would run on top of the task allocation result and would decide where and when the drone should recharge. The choice of the first implementation was due to the fact that this second option would in the first place lose the optimality achieved in the TSGA algorithm due to the fact that the algorithm would not decide the task allocation based on the bays spacial characteristics, but only based on the task spacial characteristics. As an example, imagine that a drone would almost run out of battery in the end point of a task, near another task pick up point. If this post processing strategy was to be implemented, the algorithm would immediately allocate that ”near” task, while there might be other tasks near the recharge bay, that could result in a most optimal solution. On the other hand, this option could not assure that it would be feasible to get into recharge bays during the mission completion. Having opted for the first procedure, it implies that the drone will always recharge on the nearest bay, considering that the next tasks to be performed are not know to the agent and a decision about which bay to go based on the position of the next tasks is not possible. Despite this, since all the allocation orders are being tested, it is considered that the optimality of the algorithm is conserved.

The next decision would be regarding when to send a drone to recharge, i.e, when should the drone be considered with ”low battery” and a recharge task added to the allocation. Two hypothesis were considered: the first one was the definition of a minimum threshold below which the drone would be sent to recharge. For example, every time a drone drops below 3 energy units, it would go to a recharge bay. This option clearly lowers the optimality of the algorithm. To prove this, one might just imagine a small task that would require less energy than the minimum energy threshold, beginning exactly where the drone is and in the direction of the recharge bay. Also, this option gives the chance of a drone to run easily without any energy left, for example when at a certain recharge decision point, the drone would need more energy to reach the nearest bay than the energy threshold. Considering these limitations, the proposed implementation relies on the evaluation, at each time that the a task is appended to a drone schedule, if the drone will have enough energy to perform the task that the algorithm is trying to allocate and then to go to the nearest bay (relatively to the task drop off point). If that verification is false, the drone will go recharge in the nearest bay (relatively to the point he is at). It is ensured that the drone can reach the nearest recharge bay because it was evaluated before appending this task.

To test this implementation, let us create 2 different recharge bays in the mission map, as follows in Figure 4.9, with the bays being the blue squares.

Because we have seen that with 15 energy unit battery limitation to each drone, all tasks could be completed (with no need to recharge), for this analysis the energy limit was set to 10 energy units. To use as comparison, the task allocation with no recharge and with 10 energy units limitation is represented in

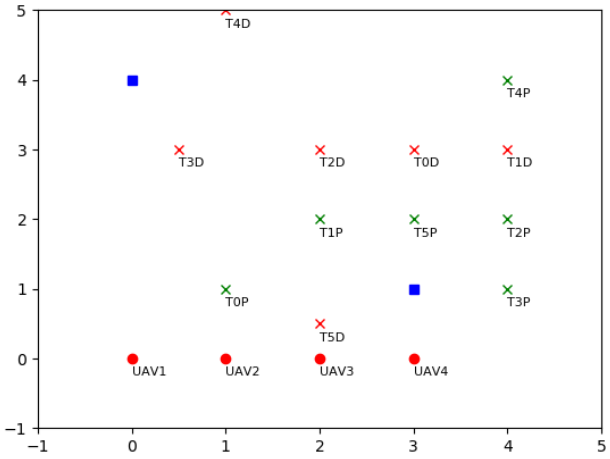

Figure 4.9: Data Set 1 environment with recharge bays

table 4.7:

Task Allocation Result

Agent	TA
1	[0,4]
2	[0,5]
3	[2,5]
4	[2,1]

Mission time: 440.957

Table 4.7: Task Allocation 5

Let us first of all mention that again task 3 is the one not completed but that the algorithm was able to reach a task allocation result with the same mission time but with a better total energy consumption compared to the task allocation achieved before the implementation of the new score function, proving that this modification improved the capacities of the studied algorithm.

With the implementation of the recharge option, the task allocation result is depicted in Table 4.8 , where ”r” represents a recharge task. Also note that the algorithm (as an implementation option) forces the drones to end the mission in a bay. This decision allows the algorithm to run many times sequentially, with different data sets and with initial battery conditions, which can be useful to perform optimizations over many time intervals of a bigger mission.

Task Allocation Result

Agent	TA
1	[0,r,4,r]
2	[0,5,r,3,r]
3	[2,5,r,3,r]
4	[2,1,r,3,r]

Mission time: 834.29

Table 4.8: Task Allocation 6

Note that Figure 4.10 only differs from Figure 4.7 in tasks 3 and 4, that are the ones for which the limited

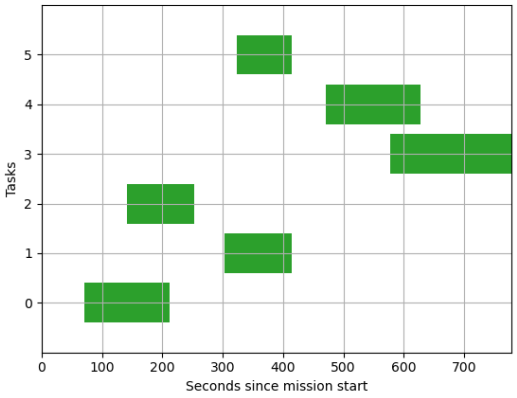

Figure 4.10: Task Execution Times w/ recharge

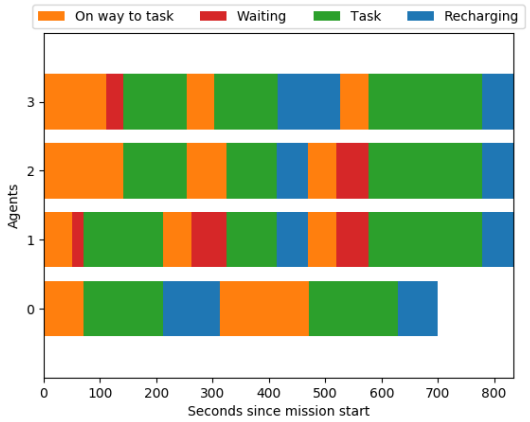

Figure 4.11: Agent Scheduling w/ recharge

energy is not enough.

To conclude the discussion, in Algorithm 4 it is possible to observe the pseudo-code of the created algorithm including the recharge procedures.

Algorithm 4 Task Sequential Greedy Algorithm with Recharge

In: *Tasks, Agents, Bays*
Out: *Task Allocation*

```
1:  procedure TSGARECHARGE(Agents, Tasks, Bays)
2:      Define all possible tasks permutations
3:      while order_of _permutations do
4:          while unallocated_tasks = True do
5:              while unsuficient_agents=True do
6:                  Find next agent
7:                  Compute energy to reach nearest bay after task      ▷ Nearest from the drop off point
8:                  if sufficient_energy do
9:                      Allocate task to next agent
10:                 else do
11:                     Allocate recharge to nearest bay                ▷ Nearest from actual location
12:                     if all_agents_tested do
13:                         Erase task from allocations                 ▷ Task uncompleted to this permutation
14:             Actualize agents info
15:             Actualize unallocated_tasks
16:         Allocate a recharge task to all agents                      ▷ End mission in bay
17:         if f(P) > best do
18:             Actualize best
19:             Actualize TA
20:     return TA
```

4.5 Algorithm Characterization

As mentioned in Section 3.2, in order to better understand the full potential of an algorithm, its scientific interest and also its applicability to real life problems, it is of extreme importance to characterize the algorithm concerning not only the resources needed to use it but also its limitations. With this in mind, in this section the performance of the modified TSGA with recharge capability will be studied, conducting

several analysis that will be developed by varying the variables of the environment and observing the run time of the algorithm. It is worth mentioning in order for other researchers to compare results, that the simulations were ran in a computer with an Intel(R) Core(TM) i7-770HQ CPU @ 2.80GHz processor, with an installed RAM of 16GB.

The first study is regarding the variation with the number of tasks and with number of agents. For these analysis, each number of tasks and agents was tested with 5 different environments, and each environment simulation was ran 10 times. The 5 different environments are tested to eliminate the possible influence of tasks and drone spacial characteristic, and the 10 runs are supposed to eliminate any occasional occupation of computer memory of processing capabilities that might influence the results. The average of the 10 runs was computed to be the time associated with the specific environment and then the average of the 5 environments was computed and represented in Figures 4.12 and 4.13, with the respective error bars, obtained through the maximum deviation from the 5 environments. This means that for each point shown in the figures below, 50 simulations were performed. This setup and approach was repeated to all variable studies in this section.

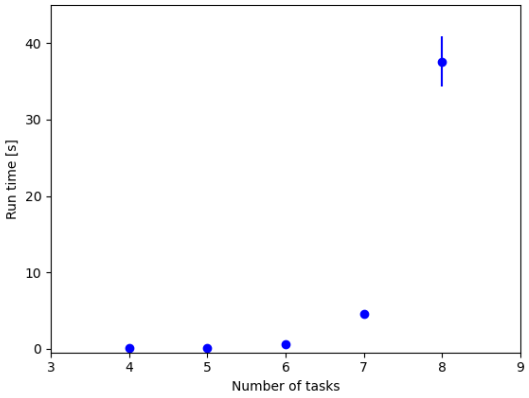

Figure 4.12: Run time variation with tasks

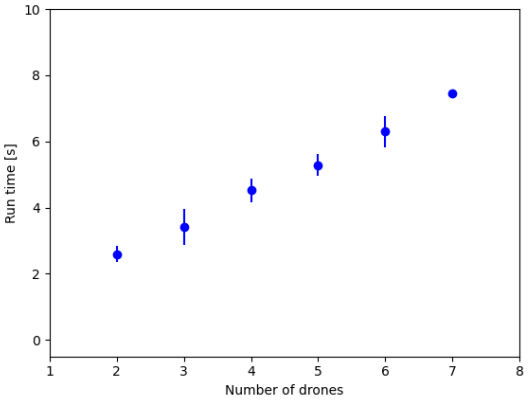

Figure 4.13: Run time variation with drones

As one is able to observe in Figure 4.12, run time rapidly grows with the increase in the number of tasks. In fact, this growth is more than exponential, it is close to a factorial growth due to the combinatorial nature of the algorithm, as expected. This is a clear disadvantage of the algorithm, given that for a increasing number of tasks, the run time is higher than the acceptable for a real life application, unless one can afford more computational power, which is something that is usually available regarding transportation companies that are someone that can be interested in the work developed in this thesis. Note that all the points have an error bar, but it is only visible for the point concerning 8 tasks due to the scale of the plot and the minimal errors that occurred in the other situations. In a matter of available drones, given Figure 4.13, it is possible to conclude that the run time of the entire procedure varies linearly with the increase of the number of drones, which does not establish a problem regarding operational implementation of this algorithm.

It is worth stating that for the study of the run time variation with the number of tasks, the number of drones was maintained constant (4 drones). Regarding the study of the run time variation with the

number of drones, the number of tasks was kept constant in 7 tasks. In both of these analysis, number of recharge bays was also constant (2 bays) and the average number of drones requested per task was 2.

The following studies were conducted with an equal procedure in order to evaluate the influence of the number of recharge bays as well as the average number of drones per task. For the analysis represented in 4.14, number of tasks was kept constant at 7, number of agents constant at 6 and agents per task was also constant with a value of 2. In what concerns the analysis of Figure 4.15, number of drones and tasks was kept and the number of bays was established in 2 bays.

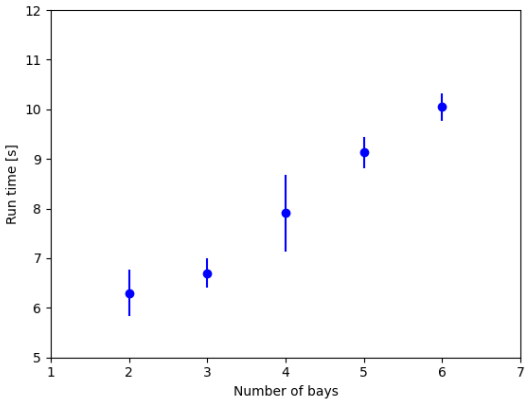

Figure 4.14: Run time variation with bays

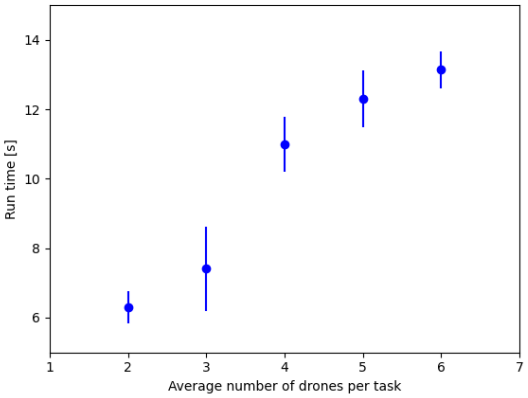

Figure 4.15: Run time variation with average number of drones per task

The analysis suggest that the run time increases approximately linearly with the number of bays in the environment. At the same time, the complexity increase with the average number of drones per task, observing the first three points, an exponential variation was expected. However, the points of 5 and 6 average drones per task registered a not so fast growing tendency. For this reason, and for the points studied, we can say that the run time increases faster for low average drones per task, with a decreasing growth as we go up in the average number of drones. This might be explained by the fact that we are reaching the total number of drones in the environment (remember that the simulations were ran with a fleet of 6 drones). It is also worth noting that the error bars of this study are the bigger ones which might suggest that this variable is the one with less correlation with the elapsed run time of the algorithm.

4.6 Case Study Simulation Results

In this section a direct application of the developed software will be presented from the perspective of a client, i.e, someone that is using the work of this thesis to plan a transportation environment. The analysis that can be conducted using the developed code are numerous. Let us consider that there is a fixed drone and task set, meaning that the geometry and spacial conditions of the problem are defined. In the first place, a user of the developed algorithm is able to study where to optimally place the recharge bays in order to maximize the time of the mission, the consumed battery of the mission or the number

of tasks performed (by tuning the coefficients of the objective function). A clear example of this is, for instance, the owner of a big warehouse where drones are transporting parcels to the adequate place inside it. There are also other analysis that can be performed as for examples some foundations to support the design of the drones or if one wants to invest in its drones, for examples in the enlarging of the drones' carrying capacity, increasing their velocity or even invest to reduce their consumption per distance unit travelled. This algorithm can be used to study which one of the investments has a bigger return, regarding the task allocation mission and the selected objective function.

In this context, let us exemplify an application of our algorithm. Let us also imagine that the following mission, with 7 different tasks and 3 different drones, as seen in Figure 4.16. This example might represent a warehouse or a factory with different drones working in the transportation of parcels or manufactured parts. The analysis that will be performed to prove the utility of this algorithm and of this type of study to the owner of the business.

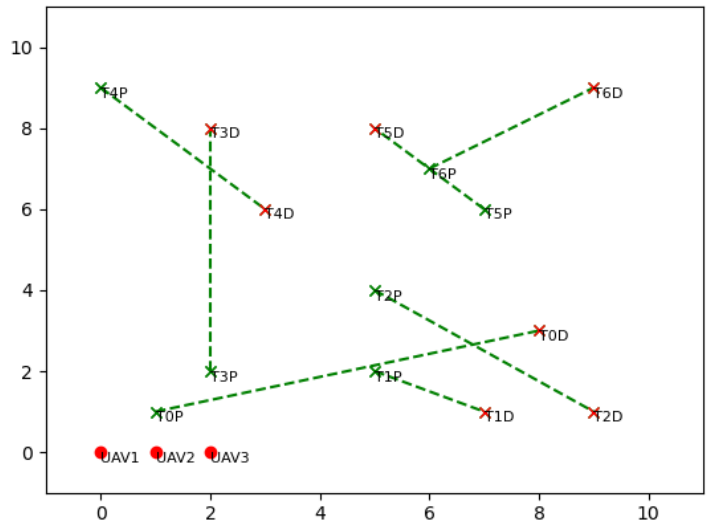

Figure 4.16: Case study 1 mission environment

The tasks were generated randomly, as well as the number of agents needed to perform each task. By this means, tasks 0,2,4 and 5 only need one drone to be carried whereas tasks 1,3 and 6 need 2 agents to be performed. It is also worth registering the characteristics of the drones: the initial positions are as represented in the image, its battery is limited to 20 energy units, the energy consumption is 1 energy unit per distance unit and its velocity is 0.2 distance units per time unit. It is also worth mentioning that the objective function is highly prioritizing the number of tasks that are being performed, giving small coefficients to the mission energy and time.

The algorithm was ran without the presence of recharge bays (and consequently without the possibility of recharge the drones) and it was noticed that the mission could not be completed as seen in Table 4.9. Note that tasks 1 and 6 could not be performed within the available energy in the drones.

Imagine now that the owner of the warehouse has the sufficient money to invest in a recharge bay, and

Task Allocation Result

Agent	TA
1	[0,2]
2	[3,4]
3	[3,5]

Table 4.9: Task Allocation 7

wants to study where to install it, inside the warehouse, in order to maximize its utility (maximize the objective function) To perform this study, 400 simulations were executed, testing every position possible for the recharge bay with a 0.5 distance units resolution, covering all the map from $x = 0$ to $x = 10$ and from $y = 0$ to $y = 10$ with 400 different points. The objective function retrieved from each one of the simulations was stored and the heat map represented in Figure 4.17 was produced with the results.

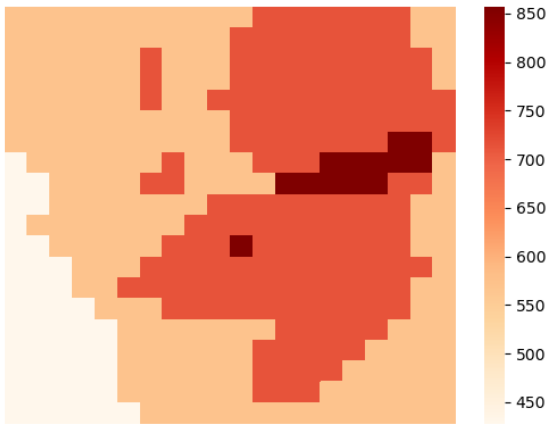

Figure 4.17: Objective value for variable positions of the recharge bay

The results are coherent with the geometry. Note that in the left down corner, where the drones begin and therefore where they have more battery available, and where there is less concentration of pick up and delivery points, the objective function earning in locating there a recharge bay is the lower on the map. On the other hand, the greater values are positioned where there is a bigger density of tasks and a single point in the center of the map.

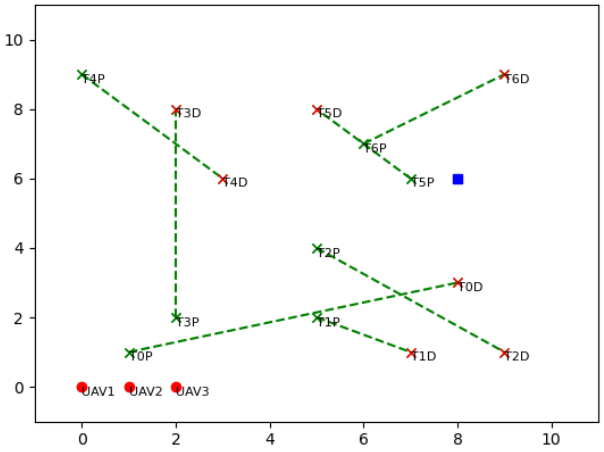

Figure 4.18: Collocation of the recharge bay

Given the heat map, one of the more optimal positions to locate the recharge bay is the point $(x = 8, y = 6)$ as depicted in Figure 4.18. Assuming that a bay is installed in this position, the task allocation of the drones is as represented in Table 4.10 :

Task Allocation Result

Agent	TA
1	[4,5,6]
2	[0,1,6]
3	[1,2]

Table 4.10: Task Allocation 8

With the implementation of this recharge bay, the system was able to perform more tasks, but was not able again to perform all the tasks, as we see that task 3 was at this time unfinished. Let us at this point imagine that the the owner wants to invest in the second recharge bay, to be able (possibly) to perform all tasks. We will now perform a similar procedure to the one done for the collocation of the first bay, this time with the first bay fixed and the new one iterating along the map. The result is shown in Figure 4.19.

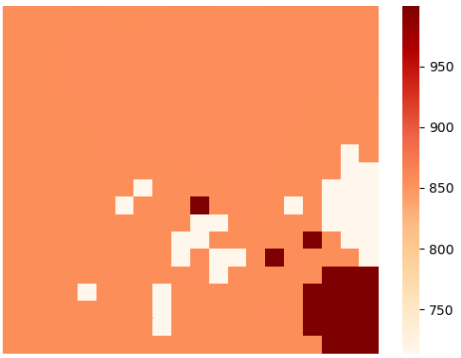

Figure 4.19: Heat map - second bay

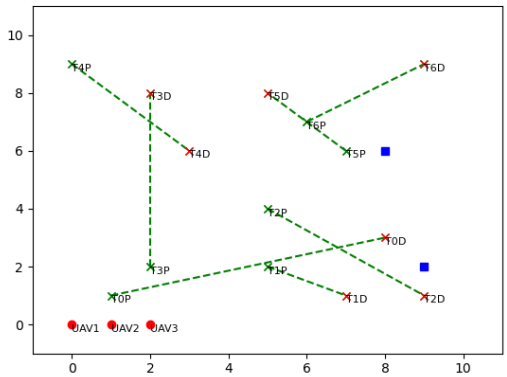

Figure 4.20: Recharge bay collocation

In Figure 4.20 one can observe the environment map with the second bay installed according to the results of Figure 4.19. Note that the heat map is in its majority uniform and the objective function has the same value as the map with just one bay. This means that at these locations the inclusion of another bay does not have any influence on the system response. The locations with light yellow coloring are the locations where the existence of a new bay (in those locations) worsen the performance of the algorithm. This happens because the decision of where to recharge is greedy, as the drone opts for the nearest bay. The dark points are the ones that enable the fleet of drones to perform better and conclude all the tasks. With this second bay, the task allocation result is presented next, in Table 4.11 as well as the graphic representation of the schedules of agents and tasks.

Task Allocation Result

Agent	TA
1	[3,5,6,1]
2	[0,1]
3	[3,4,6,2]

Table 4.11: Task Allocation 9

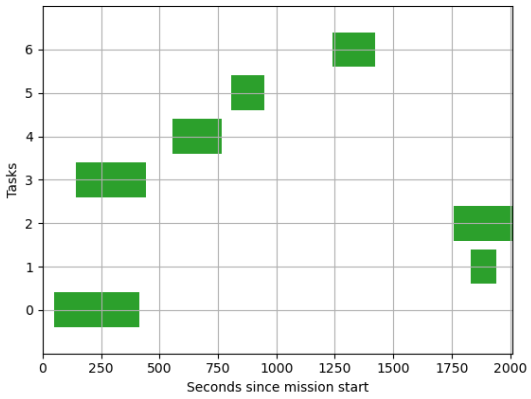

Figure 4.21: Task schedule

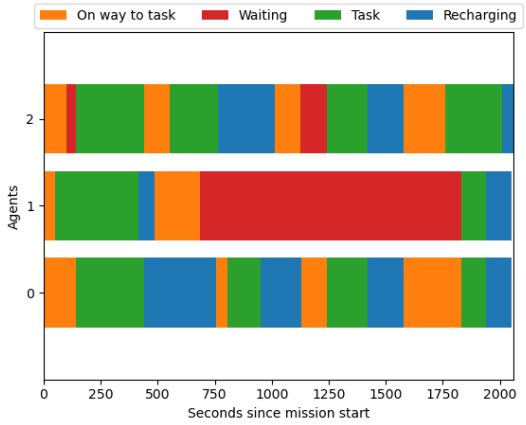

Figure 4.22: Agents schedule

Note that all tasks are performed. For future comparison, it it worth noting that the total mission time having the two recharge bays installed is 2059 time units and the total energy consumed is 97.32 energy units.

Imagine now a new scenario where the owner wants to decide whether to invest in two different modifications on the drones. The first one enhances the velocity of the drones 25%, but the energy consumption also grows 5%, or another project (project 2) that also enhances the velocity in 50% of the drones but jeopardizing their carrying capability to a point where all the tasks need one more agent to be performed. The developed algorithm also enables the owner to make a complete analysis in what concerns mission time, energy consumed and tasks completed.

Both the projects impact the conclusion of the entire set of tasks, as in project 1 task 3 is unperformed and in project 2 task 0 is unperformed, given the spacial characteristics of the problem and the position of the bays (that might now not be optimal given the new characteristics of the problem). For this reason, in

Task Allocation Project 1

Agent	TA
1	[4,6,1]
2	[0,5,6]
3	[2,1]

Mission time: 1468.65
Total energy: 87.22

Table 4.12: Task Allocation 10

Task Allocation Project 2

Agent	TA
1	[3,5,6,2,1]
2	[3,4,5,6,2,1]
3	[3,4,6,1]

Mission time: 1827.42
Total distance: 141.85

Table 4.13: Task Allocation 11

the perspective of the owner, keeping the main objective of performing all the tasks, none of the projects is a good one.

Chapter 5

Task Allocation with Relay Maneuvers

In this chapter, relay maneuvers will be studied, implemented and ultimately incorporated in the complete task allocation algorithm. As noted in Chapter 2, relay maneuvers in a parcel delivery environment are not widely studied in scientific literature, and for that reason, a significant part of this chapter work was not only formulating the problem in a coherent and relevant way such that the results serve the expected and planned purpose given the cooperative task allocation study of this thesis, but also generating an interesting systematic approach to solve allocation problems with relay maneuvers.

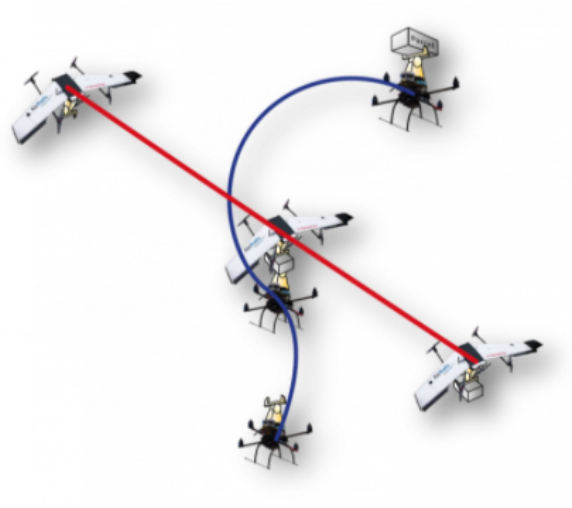

Figure 5.1: Graphical representation of a relay maneuver

In the first place, it is relevant to define what is a relay maneuver in the context of the studied problem. As stated in the introduction, this thesis is a result of a research work performed in the context of REPLACE project which has the main goal of studying a fast drone parcel delivery environment. Given this objective, it is mandatory to study cooperative tasks as a consequence of the physical and real limitation of each one of the drone such as endurance. For this reason, an explicit study of a in-flight process in which drones are capable of transfer parcels between each other in all its aspects (control, planning, mechanics) is one of the main objectives of this project. In this context, a relay maneuver is defined to be the transmission of a parcel between two drones in an in-flight procedure, as seen in

Figure 1.1 or 5.1. The ability to perform this maneuver enables a new degree of freedom regarding the cooperation between agents.

5.1 Problem Statement

Given the definition of relay maneuver, the main goal of this chapter is to include relay maneuvers in the task allocation algorithm. With the relay points, the tasks are from now on considered to be composed by various sections that will be processed as separated tasks. For this, some considerations should be taken into account.

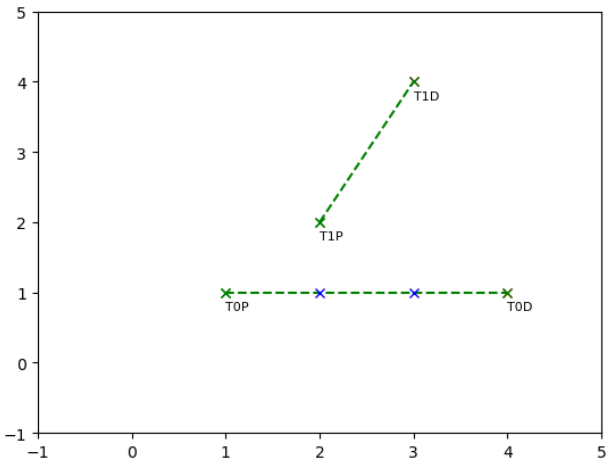

Figure 5.2: Example of environment with relay points

In Figure 5.2 we are able to see 2 different tasks. Task 0 is divided in 3 different minor tasks by the means of 2 relay points whereas task 1 does not have relay points and thus, is not divided.
In order to ensure language coherence, let us detail some useful definitions. From this point on the total task will be called *task* (green dashed line in the graphical environment), and the referred minor tasks will be named *relay task*. A *relay point* is the place where a relay maneuver is set to happen (represented with blue crosses in the graphical environment). A more rigorous definition of *relay task* can be: a relay task is a task limited by at least one relay point. Additionally, to enumerate and refer the relay tasks, a new notation will be used. As noted, all relay tasks are associated to a original ”big” task, and the notation used will be ”X.Y” where ”X” relates to the original task associated with the relay task (global numeration) and ”Y” represents that the relay task is the Yth inside of the original task (internal numeration in each original task). As an example, observing Figure 5.2 the relay task that begins in the pick up point of task 0 and ends in the first relay point has the numeration of ”0.0”, the ”middle” relay task in task 0 is referred as ”0.1” and the last relay task that begins in the second relay point and finishes in the task drop off location will be designated ”0.2”. Task 1 (with no relay points) will be designated ”1.0”,
In order to better formulate the problem, some assumptions should be made:

1. In the first place, it is considered that 100% of the performed relay maneuvers are successful, and

for that reason no parcel is lost and all maneuvers require the same operational time;

2. It is also assumed that relay maneuvers are only performed in tasks that require only one agent; this consideration does not come with the intention of decreasing the complexity of the problem but instead some increasing the real applications of the study since it would be very difficult to create the transmission mechanisms for transferring a parcel carried by more that one drone;

3. It is also supposed that for a relay maneuver to take place, the receiving drone should reach the relay point earlier than the drone that is providing the parcel (or simultaneously); this is assumed to ensure that the first drone does not have to wait for the second one knowing that this could cause some problems to the system.

Regarding this last assumption, let us consider the example of a drone carrying a parcel almost running out of battery that ”calls for” a relay maneuver in the limit of its endurance (the ”calling” of the relay maneuvers will be extensively discussed in Section 5.2; the fact that it would have to wait hovering for the receiving drone could jeopardize the optimality and even the feasibility of some approaches and solutions; for this reason, it is established that the algorithm must be programmed in a way that the operation in the relay point should be performed as soon as the carrying drone arrives at the point. It is also understood that it will not be possible to force the drones to arrive exactly at the same time at the maneuver location, and for this reason, the drone that is supposed to wait is the drone that is going to pick up the parcel and not the one carrying it.
Given these assumptions, the optimization problem relies on finding the best task allocation for each drone, similarly to the Chapter 4 problem. The first strategic approach was to split one integral task into more than one task, separated by one relay point, as it is possible to observe in the Figure 5.2. How to split the tasks and how to process the algorithm will be discussed in the next section.

5.2 Problem Analysis

In this section the strategy to approach the problem will be discussed in a top down sequential strategy. This means that the problem will be analysed firstly as a whole, taken into consideration general requirements and decisions, and just then more particular issues will be tackled. It is worth recalling the inputs that the algorithm is expected to receive that are only a set of tasks and the characterization of the drones, meaning that the user is not expected to define the relay points. As a consequence, the algorithm is expected to (optimally) define the relay points that it will be used for each one of the tasks (if any) according to the drones limitations and the specific geometry of the mission. This happens because the goal of the algorithm is to be as general and useful as it could be to the user that will not have to think and decide where to define the relay points.

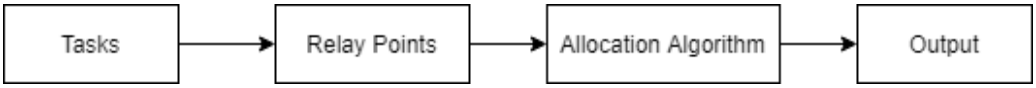

Figure 5.3: First approach representation

We can observe in Figure 5.3 one sequence of decisions and information flow that represent the referred problem. With this strategy to approach the problem, three main challenges arise.

The first one relates with the time synchronization of the drones in the relay point, remembering the third assumption we considered in the past section (the receiving drone is expected to be at the relay point no later than the delivering drone). This demand is easily tackled by some manipulation on the coding of the problem, for example making sure that the second drone has enough time to reach the relay point, or otherwise assign the first drone a waiting time for it to be possible to achieve by the second drone.

The second challenge that arises is resultant of the algorithm decision of where to locate the relay points as well as how many relay points in one single task, while a related third one is, given the relay points, when to trigger or ask the relay maneuver. We will further go through this 2 problems together as they are intrinsically interrelated. These challenges, due to the difficulties to the solution that they impose, were considered to be the core of an algorithm with relay maneuvers and where many of them can have significant contributions. A lot of time and thoughtfulness was needed to fully analyse the numerous implications and consequences of each of the decisions taken in this context. It is worth referring again that the algorithm was first and foremost kept generic with the goal of achieving the best optimality possible.

The first idea was to ask for a relay maneuver in the exact point where the carrying drone would have enough energy to reach the nearest recharging bay. But how would the programmer confirm that that was the best option given the whole task set and the total geography of all the tasks? How could the programmer know that it was not better to perform the relay a little bit early because it could save time to the total mission, enabling a drone to reach another task or a recharging bay earlier? Let us imagine for example an situation where the recharge bay is located near the parcel trajectory, as seen in Figure 5.4.

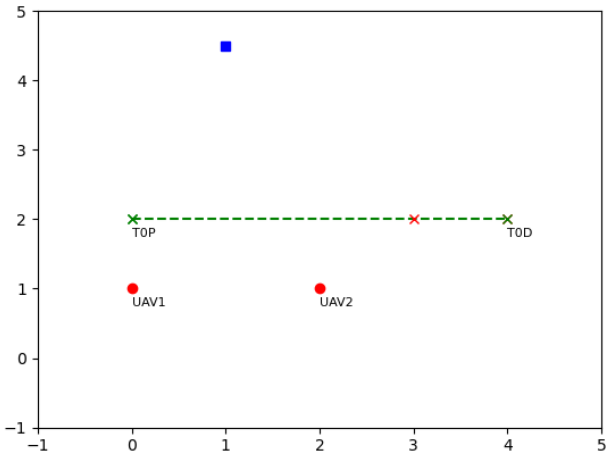

Figure 5.4: Case study map 1

Let us imagine for the sake of this argument, that this task is in the middle a bigger set of tasks and that the point where drone 1 would have just enough battery to reach the recharge bay (blue square) is

the red cross. If we assume this strategy, the red cross becomes the point where the relay maneuver will take place. At this point, I would ask the reader to take a second and think if this is, for this small problem, that decision is the optimal one.

It is not. In the Figure 5.5 it is possible to observe, represented with a blue cross, another option of a relay point that would save the entire distance from the blue cross to the red cross, diminishing the total distance travelled by the coalition of drones. Note that in the first strategy, drone 1 is set to travel from its initial point to the point T0P (task 0 pick up) and then to the red cross, proceeding next to the recharging bay while drone 2 is set to travel from its initial point to the red cross and then to point T0D (task 0 drop off). On the other hand, if the relay point would be the blue cross, drone 1 would travel only until the blue cross and proceed to the recharge bay from this point. This small difference might seem almost negligible but throughout dozens of relay decisions it has a great impact in consumed energy as well as in mission total time, noting that with the blue cross drone 1 would be available to be considered to another task before than with the red cross. It is also worth noting for the validation of this argument, that if the drone has enough energy to go to the red cross and to the recharge bay, it is also guaranteed that it has the energy for this new trajectory.

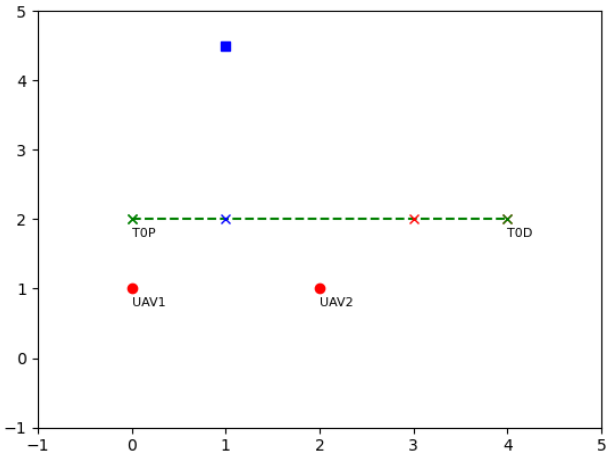

Figure 5.5: Case study map 2

Concerning this environment and assuming that the total mission time and energy are goals of our optimization (as in the previously proposed strategy), we can conclude that the decision to perform the relay maneuver according to this strategy is not optimal and its implementation would limit the capability of the algorithm to reach an optimal solution. This means that this is not the way to overcome the two outlined problems.

Having another look to this specific task map, another idea is setting the relay point to be the point of the trajectory that is closer to the recharge bay, as the blue cross in Figure 5.5. This would give a ”virtual” optimal solution to the presented environment, as the relay point would be exactly the proposed one, used to prove the non-optimality of the first approach. With this modification, the travel distance from the relay point to the recharge bay would be the minimum possible and the coding behind would not be

very complex.

However, after some testing, it is not hard to conclude that this is also not a good solution for 2 main reasons. The first one is related with a problem described before: if the relay maneuver is not called only when the drone is running out of battery, the question relies on what should be the occurrence that forces the relay maneuver to happen. The program could easily foresee if the drone is below a defined threshold of energy and if so, call the relay maneuver but this would also decrease the optimality of the algorithm as we discussed in Chapter 4.

Additionally, there are situations in which this point is not even the optimal, when other tasks are taken into consideration. As an example, let us run an analysis on the following map.

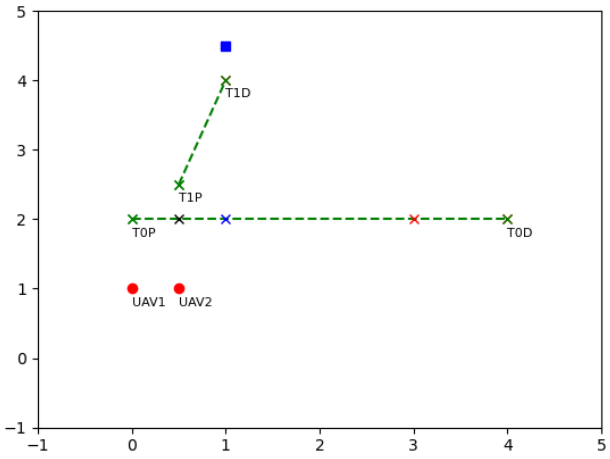

Figure 5.6: Case study map 3

In this case a new distinct task is added to the referred example. Let us assume for simplification of the argument that it has no relay point. In respect to the last strategy, the relay maneuver would occur in the blue cross. Note that in this case, both tasks need to be performed. In a similar argumentation, we are able to observe that the relay point position that minimizes both the travelled distance of both drones and the total mission time is no longer the blue cross, but is now the black cross instead, once it is the nearest point to the pick up point of task 1 and there is no waiting time due to the initial position of drone 2.

With these examples, and after thinking about the implementation and correctness of some other strategies, the conclusion is that it is difficult to establish *a priori* a generic rule, or chain of rules or even a deterministic strategy that would be successful tackling the two aforementioned problems, given that deterministic algorithm would need to decide how many relay points in each task, the position of the relay points and the reason to call the relay maneuver. It would be extremely challenging to build a deterministic algorithm that would ensure that the decision making was the best for the global mission (one is able to imagine a mission with a big number of tasks) and not only to a small system near to the point where the decision is being taken. For this reason, a non-deterministic approach will be implemented in this topic.

5.3 Proposed Solutions

Given the above discussion, we will consider two different non-deterministic approaches that, after some research referred on Chapter 2 and exploration about non-deterministic optimization, heuristics and meta-heuristics, seemed good strategies to this specific problem.

1st Proposed Approach and Algorithm - Geographic Search Heuristic with Memory

One idea that was thought to overcome the aforementioned problem of the deterministic approach was an heuristic where each drone would have a ”vision radius” where it would see every task pick up point, parcel trajectory or recharge bay that would exist in that radius. This window would be considered by the drone as all the option it would have to choose. The objective function would be defined for example as the fraction of travelled distance that the drone would actually be carrying a parcel, and each drone would have the goal of maximizing this function.

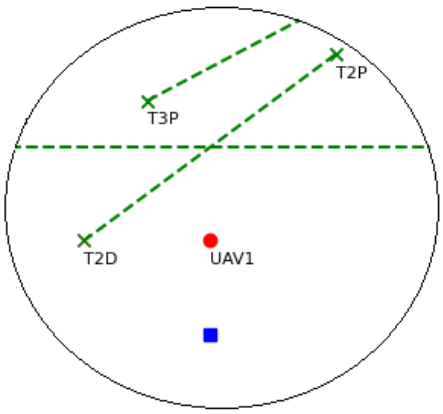

Figure 5.7: Drone vision radius example

Regarding a first iteration, the decision of where to go would be random and the drone would store in memory which zones of the entire map would result in a higher reward for itself. The algorithm would have a finite number of iterations and the decision making of the drones would be tuned up, from iteration to iteration based on its memory, in a way that it would have a greater probability of choosing the zones with higher rewards, while always maintaining a certain probability to other map zones in other to seek for all the task and remote tasks that could exist.

Some details of the algorithm would have to get a more in-depth study but this idea is the core of this proposed strategy. However, some difficulties and problems regarding this algorithm were shorty noticed. The programming behind this solution would be heavy and really test demanding, once there would be too many unforeseen and unpredictable situations and circumstances, as for example how to proceed when the drone did not have anything in its vision radius or defining the stopping criteria of each iteration. Also, there would be no assurance that the solution would be optimal or even close to optimal. Extensive simulations would have to be ran in order to achieve some conclusions about the optimality of

the solution.

Furthermore, the big step of decentralizing the algorithm was a relevant turn down to proceed with this algorithm, not only because of the baseline research that needed to be done, but also because no work developed until this point would be used, and the environment, framework and programming would need to be done from scratch. Also, we would not be able to compare the results with the ones achieved in the previous chapter.

2nd Proposed Approach and Algorithm - Relay Point Binary Optimization

Given all the above challenges and down sides presented regarding the last algorithm, another hypothesis is formulated. This time, the main focus is at the one hand to maintain a centralized algorithm and on the other hand, create a possibility for the utilization of the work performed in Chapter 4.

As concluded before, the two decisions of where and when to perform the relay must be found by the algorithm. However, it is possible for the programmer to give some possibilities of relay points locations to the algorithm and these two decisions would be taken on top of those possibilities. This is of course a relaxation of the problem and its constrains, narrowing the possible locations of the relay points, and reducing the optimality of the result. However, the programmer is in practice creating a grid, or a mesh, on top of which the algorithm is going to optimize. This means that the quality of the solution can be increased with the refining of the grid as long as computational power is available.

This strategy would be programmed to be solved using binary optimization algorithms. For this, after the definition of the possible location to the relay points, associated with each one of those points, a binary variable would be created, encoding an active or inactive points. The decision variable vector would be the vector made of all these binary variables to be set by the binary optimization algorithm, resulting in the best places in the map for performing a relay maneuver.

Inside each iteration, with a fixed decision variable vector and thus, with fixed locations to the relay maneuvers, an allocation algorithm would decide each drone task allocation and would output a general score of the task allocation, that would be used inside the binary optimization of the relay points. A diagram representing the idea of this algorithm is presented in Figure 5.8.

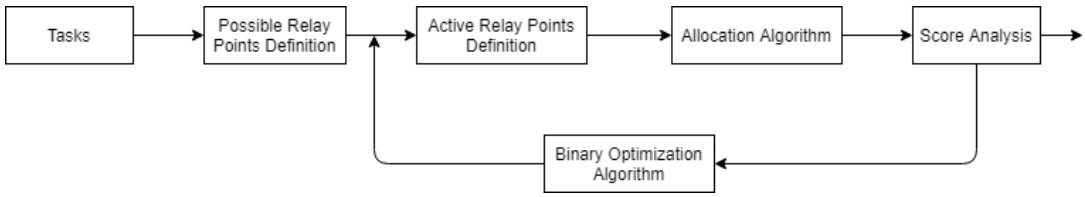

Figure 5.8: Relay Point Binary Optimization Diagram

This approach would ensue numerous advantages. Considering that there are several well known and studied meta-heuristics, the binary optimization algorithm would be an available one, with recognized performance metrics. For that reason, the implementation would be more controlled and the performance of the algorithm better evaluated. This approach also enables the use of some allocation algorithms studied in the last chapter (with the necessary modifications for example to ensure time synchro-

nization). We will in the next section discuss the applicability of TSGA for example, its limitations, pros and cons when applied in this context.

For the study to be conductive to an appropriate solution, the programmer should run some simulations with different possible relay points (considering the available computational power) in order to understand if the task division is not limiting the optimality of the solution. For instance, let us imagine that the programmer splits one task, i.e creates possible relay points, in 4 different paths, as seen if Figure 5.9. It is clear, that if the programmer decides to split the task in 8 different paths, the result will be at least as good as the one with 4 paths. In fact, the 8 divisions, contain the 4 different divisions, and so, if the optimal solution is performing a relay maneuver within the 4 relay points in Figure 5.9, the algorithm running with the 8 possible relay points (Figure 5.10) will also find that solution, once the space of solutions of the algorithm with 8 possible relay points contains the solution space of the algorithm with 4 possible relay points.

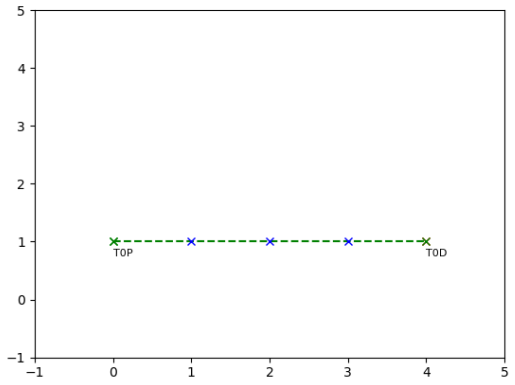

Figure 5.9: Task w/ 4 paths

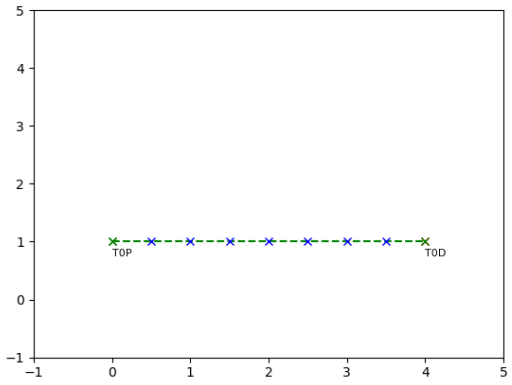

Figure 5.10: Task w/ 8 paths

The other way around is not valid, and this is why a greater computational capability may (it is not certain) conduct to a better overall result, considering that the created ”mesh” has more possible points. In the limit, if one would split the tasks in an infinite number of path, the algorithm would be running almost on a continuum space, testing all possible positions, instead of being a discrete optimization.

Because of all the positive aspects outlined about this second approach, this will be the one to be followed and implemented. And thus, the next analysis should be if the discussed TSGA algorithm fits our purpose in this problem. With some modifications that will be further discussed in the Implementation section, TSGA can provide a feasible and good solution regarding optimality as he have seen in Chapter 4, and so there is no reason to develop a new algorithm, at this point of the analysis. Despite this, it is also assumed that there may exist some unpredictable characteristics or problems regarding this formulation, and so an assessment of the quality of the implementation and of the fitness of this allocation algorithm needs to be performed. A probable concern is the heavy increase of the number of tasks because of the task division, that combined with the combinatorial nature of the algorithm might arise a problem in its implementation.

5.4 Computational Implementation and Discussion

In this section, the implementation of the Relay Point Binary Optimization will be discussed together with some particularities of the coding of this problem.

First of all, an important assumption was made: for one task, with m relay points, the same drone is not able to perform more than one relay task, i.e, more than one segment of the task. Recalling for example image 5.9, one drone is not able to perform the path from the pick up point to the first relay point and after, picking up again the parcel in any of the ensuing segments. This decision was taken bearing in mind that it is considered that the velocity of the drone is constant (with or without parcel). This implies that the drone that transported the parcel from the pick up point to the first relay point (regarding Figure 5.9 again), in order to be able to be on time in another relay point forward, would have to assume the trajectory of the task itself (the straight line) because the other drones, with the same velocity, will carry the parcel in that trajectory. Note that ”being on time” means that the reaching drone will not have to wait for another drone in the relay point in order to perform the relay maneuver. Thus, it would not make sense in a way that if the drone is doing the parcel trajectory itself, that drone should be carrying the parcel and the relay maneuver should not have happened in the first place.

This decision has a considerable consequence which is that the number of drones of the mission, should be no lower than the maximum number of relay segments in a single task, or in other words, should be higher than the maximum number of relay points in a single task. For example, if a task has 9 relay points (and so 10 segments in the trajectory), there should be at least 10 different drones in the environment.

Also note that the operational time of the relay maneuver is not being considered in this argumentation, meaning that if we assume that the act of changing the carrier of the parcel would imply a ”time loss”, the argument of the straight line trajectory aforementioned is not as precise. Despite this, considering that in a mission like this the goal will be always to minimize the time, and so, the operational time of this maneuver, this approximation is considered a reasonable one, noting that the complexity of the problem is not lowered by this decision.

After this small preamble, it is relevant to discuss the allocation algorithm inside the logic represented in Figure 5.8. The main idea, taking advantage of the understood and proven TSGA algorithm studied in Chapter 4, is to implement this optimization procedure using that algorithm and the knowledge acquired in its study. The main concern is the combinatorial nature of the algorithm and its scalability. However, considering an entire mission, this complexity is dependent of the number of tasks, as we have observed earlier. The focus will be changing the allocation procedures in order to meet the restrictions of this problem such as the time synchronization of the drones in the relay points without augmenting the complexity at the level of the TSGA cycle.

We will, similarly to the followed line of thought in Chapter 4, begin to implement the functions in small mission environments, and increase the complexity, making sure that the implemented functions are tested and without flaws. For this reason, the first approach to this implementation will again assume that the drones have infinite battery and therefore no recharge tasks or recharge bays will be taken into consideration.

So, the first analysis will focus on a simple case of a single task with 2 relay points with 3 drones, as shown in Figure 5.11, where a single task with 2 relay points and 3 UAVs in their initial positions are represented. For the analysis and comparison of time of the missions, let us assume that the speed of the drones is 0.02 distance units per time units. As referred before, the algorithm and the entire software does not infer any units of time or distance, meaning that it is generic and is able to run with the characteristics that the user or the programmer imposes. However, the thinking behind the units was that one unit distance represents one kilometer and one unit time represents one second, meaning that 0.02 distance units per time units is equivalent to 20 meters per second.

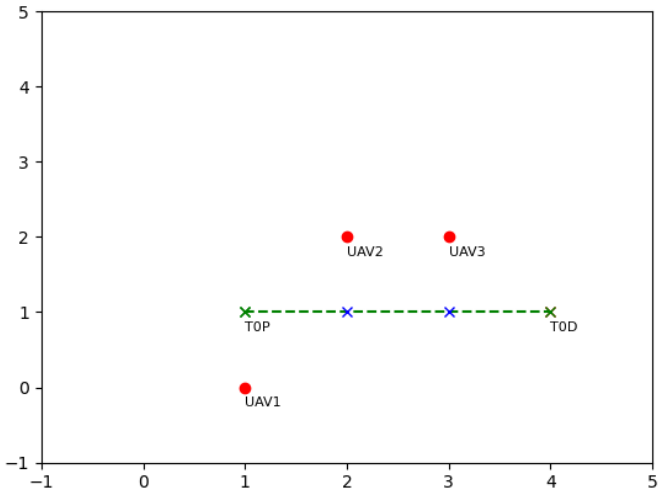

Figure 5.11: Relay development environment

In a first approach to create the algorithm that will dictate which drone is going to perform each one of the segments of the tasks, a greedy procedure was implemented from the beginning to the end of the task. This means that the procedure is to sequentially look for the closest drone for the relay task and allocate that segment to that agent. In this specific example, the algorithm will proceed as follows: analysing the pick-up point, the nearest drone is UAV1, and so the algorithm will allocate this drone to this relay task from the pick-up point to the first relay point. After, it will analyse what is the closest drone to the first relay point that is not yet allocated to another segment (does not make a different in this case). This drone is UAV2. Before allocating this agent to this relay task, the algorithm confirms that UAV2 will arrive at the relay point before UAV1 carrying the parcel. If so, ir allocates the relay task to the agent. The same procedure is then applied to relay point 3, being UAV3 the chosen one.

In this case, it is clear that both UAV2 and UAV3 will arrive to the relay points before the predecessor drone. Let us take the example of the first relay point: UAV2 has to travel 1 unit distance to arrive to the relay point, while UAV1 is set to travel 2 distance units, one to reach the pick up point and another one yet inside the parcel trajectory. For the second relay point the analysis is again analogue. The task allocation result is in Table 5.1, remembering the notation defined in Section 5.1

It is worth noting that the mission time is 200, once the UAV1 needs 50 time units to reach the pick

Task Allocation Result

Agent	TA
1	[0.0]
2	[0.1]
3	[0.2]

Mission time: 200

Table 5.1: Relay Task Allocation 1

up point (considering the referred velocity) and that each one of the segments take 50 unit times to be performed.

At this point the algorithm is set to retrieve a message of error if at some stage there is no drone in the fleet that is close enough to the relay point to arrive in time for performing the relay maneuver. For example, observe Figure 5.12: if UAV3 initial position is the point (x=3;y=5), distancing 4 units from the relay point, it reaches the point in time for the maneuver. If this happens, the algorithm retrieves an error message and erases every relay task associated with this task from all drones schedule. This means that the task is not concluded which has an associated penalty in the objective function, and for this reason, when performing the overall optimization algorithm, the system avoids this from happening.

However, given the globality of the problem, it is quickly understood that this was not a correct approach, since the purpose of the algorithm is guaranteeing that the tasks are performed, when there is not a cause that makes it impossible for the system to do so. With the strategy described in the previous paragraph, many tasks would stay unperformed because as the drones would add tasks to their schedule, more and more situations would happen where the drones would have to wait for each other which would result in an error message as described.

For this reason, and in order to ensure that the third assumption in Section 5.1 is accomplished, the idea is that the best drone to each point is chosen (both relay points and the task pick-up point), but the task execution is only deployed (the first drone will only begin the task) at such time that all the sequential drones will arrive at their determined relay point just on time. In practice this means that the algorithm will discover the ”limitative agent” and the task will only begin when that agent (that is already moving towards it point) is close enough to reach its point on time. Let us imagine the situation described before, with UAV3 at (x=3;y=5) in the beginning of the simulation, as represented in Figure 5.12.

Note that if the task is deployed as soon as the simulation begins (t=0), UAV2 would arrive with the parcel at the second relay point at t=150, while UAV3 can only arrive at the relay point at t=200, which would result in a violation of the third assumption. To go around this problem, the algorithm is set to compute a vector of the arriving times of each one of the drones to the respective relay point. After this, calculating the time it would take from the task deploy until the parcel reaches the relay point, it is possible to understand what drone is the limitative one. Let us again analyse the environment represented in Figure 5.12: the algorithm will compute the time that each drone takes to arrive at its relay point and will obtain the following vector:

$$[50, 50, 200]$$

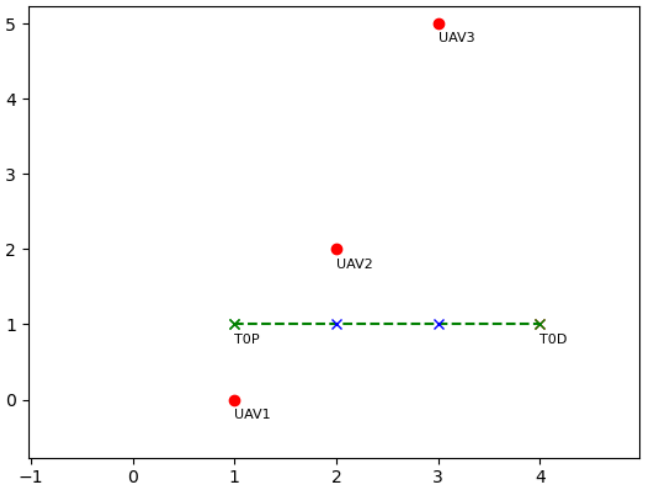

Figure 5.12: Far UAV example

meaning that agent one takes 50 time units to reach its relay point, agent 2 takes also 50 units and agent 3 takes 200 units. After this, the algorithm computes how long does it take to initiate a segment after the mission is deployed, this is, after the UAV attributed to the pick-up task starts moving. It will therefore generate the following vector:

$$[50, 100, 150]$$

By subtracting the first vector to the second, the maximum value represent the restrictive agent, and the value of the difference is exactly the time that the first drone need to wait to deploy the mission. The elements of the subtraction vector that are not negative will have to start moving before the task deployment. The ones that are zero will have to start moving no later than the mission deployment and the negative elements can start moving only after the task is deployed.

Analysing this example step by step: the algorithm will find that UAV3 is the limitative agent (as expected) and that the time difference is 50 units. So, at t=0, UAV3 will start moving towards its relay point. At t=50, UAV3 will be at the point (x=3;y=4). At this point, it needs 150 time units to reach its relay points, which is exactly the time that is needed between the moment UAV1 starts moving and the moment the parcel reaches the second relay point. Given this, this is the correct time to deploy the mission, the algorithm will ”give that order”, and all the drones will start moving. At t=100, UAV1 will be picking up the parcel and UAV3 is at (x=3;y=3). At t=150, the first relay maneuver is to be performed and UAV3 is at (x=3;y=2), and at t=200 the last relay maneuver is going to be performed. The parcel reaches its delivery position at t=250, resulting in the task allocation result represented in Table 5.2.

With this modification the algorithm will not leave any task not performed, as long as the unlimited battery of the drones is kept, and no feasibility condition is broken (having more relay points than available drones, as an example), representing a significant improvement of the algorithm implementation.

However, with this purely greedy approach, optimality is not achieved and is even highly jeopardized, as

Task Allocation Result

Agent	TA
1	[0.0]
2	[0.1]
3	[0.2]

Mission time: 250

Table 5.2: Relay Task Allocation 2

one can observe in the following example. Let the initial distribution of the drones be as in Figure 5.13.

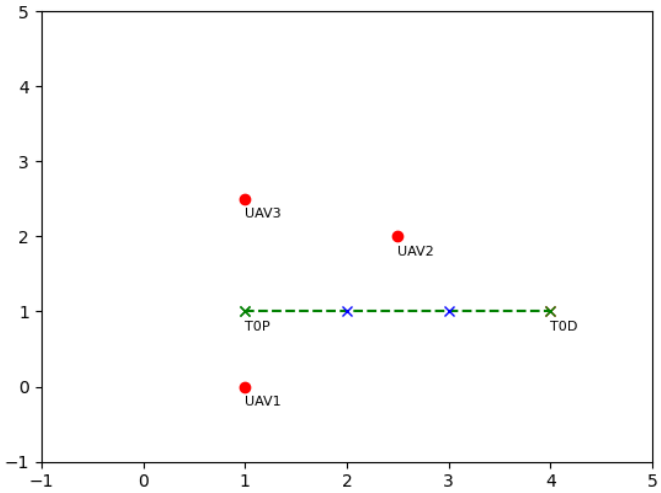

Figure 5.13: Example of non optimality of current algorithm

Following the algorithm chain of thought: regarding the pick-up point of the mission, the closest drone is UAV1, meaning that it would allocate UAV1 to that segment. After, regarding the first relay point, the nearest drone is UAV2. Referring the second relay point, the nearest drone is UAV2, however, it is already in use for the second segment, meaning that the algorithm will discover the nearest drone that is not already in use which in this case is UAV3. Given this, the result of the task allocation is represented in the Table 5.3. Note that it is also showed the total distance covered by the coalition of drones which is important for the optimality of the solution not only concerning mission time but also energy consumed.

Task Allocation Result

Agent	TA
1	[0.0]
2	[0.1]
3	[0.2]

Mission time: 200
Total distance: 7.618

Table 5.3: Relay Task Allocation 3

However, one can observe that it is possible to achieve a solution that delivers an equal mission time of 200 time units with a lower total distance travelled by the agents, which implies a reduced cost if we consider both the goals of minimizing time and consumed energy. This solution would be allocating

UAV3 to the first relay point (second segment of the task) and UAV2 to the second relay point (third and final segment). Note that concerning elapsed time this allocation would not modify the result as UAV3 reaches the first relay point before UAV1 with the parcel and UAV2 reaches the second relay point also before the parcel. In any case, it ”saves” energy. Note that the distance between UAV2 and the second relay point is the same as the distance between UAV2 and the first relay point. Noting also that UAV3 is closer to the first relay point than to the second one, it is easy to observe that this new result improves the energy consumed by the team of drones.

To achieve this result based on the conceptualized algorithm, some changes must be implemented. The algorithm will, from now on, before allocating the relay tasks to each drone, choose the best coalition, i.e the group of drones, to perform the task. This is going to be done by choosing the closest available agents to each segment of the task, forming the coalition. Inside this coalition, an optimization procedure will be run: the algorithm will test all permutations of the chosen agents to each relay point. Regarding the past example, the algorithm will test 6 different orders of the drones (all possible permutations for the drones to each one of the segments) and choose the best allocation, by the means of a minimization of an objective function:

$$f_1(\mathbf{a_k}) = k_1 \times s_{time}(\mathbf{a_k}) + k_2 \times s_{energy}(\mathbf{a_k}) \tag{5.1}$$

It is worth noting that the user can tune the k factors in order to give prevalence to any of the components of the objective function. If one tune k_2 to zero, the result will be the one from the greedy approach. It is also worth mentioning that the inclusion of the chosen coalition instead of testing all the existent drones reduces the computational effort of this optimization.

In fact, in the coalition formation it is sure that the closest agent to each point is included even if it is the closest agent to more than one point (as it happens in the past example). This guarantees a good achieved optimality because we eliminate the influence of the order by which the points are analysed to chose the best agent, in a first level. The only case that this decision threatens optimality is if there is a second closest agent to a point that will not be considered in the coalition because the closest agent itself is still available. Let us take the case of the past example. The achieved solution would not be optimal only if there would be a forth drone in a position where it would be farther away from the second relay point but closer to the first relay point than UAV3, being at the same time farther way from the first relay point that UAV2. This would lead to a solution where the allocated drone to the second segment is not the second closest to that point because that drone is not considered in the first place noting the coalition is not formed by this drone because when analysing the second segment of the task, the closest drone is still available to be part of the coalition.

This optimization level, where within 1 single task the best coalition of drones is chosen and allocated to the relay tasks will be called *micro allocation cycle* from now on to denote the difference to the optimization cycle that will be discussed next.

After the implementation of this change in the algorithm, the allocation result is in Table 5.4. Note the difference in total distance from the result achieved before this optimization cycle.

Task Allocation Result

Agent	TA
1	[0.0]
2	[0.2]
3	[0.1]

Mission time: 200
Total distance: 6.920

Table 5.4: Relay Task Allocation 4

After this change, the algorithm is able to handle the allocation within a single task. The next step in complexity is adding more than one task. The algorithm is made in a way that it is not mandatory for the tasks to have relay points, but it functions with as many relay points as the user wants (limited by the number of drones as it is mentioned before). In algorithm 5 it is possible to analyse the pseudo-code of *micro allocation cycle* procedure.

Algorithm 5 Micro Allocation Cycle

In: *Task, Relay Points, Agents*
Out: *Ordered Coalition*

procedure MICRO ALLOCATION CYCLE($Task, RelayPoints, Agents$)
  while *Relay_Tasks* **do**
    Find nearest agent and append to coalition ▷ Coalition Formation
  Define all coalition agents permutations
  while *untested_permutations* **do**
    while *Relay_Tasks* **do**
      Allocate next agent in permutation to next relay task
    Compute $f_1(a_k)$
    if *f1(ak)* $<$ *best* **do**
      Actualize *best*
      Actualize Ordered Coalition
  return *Ordered Coalition*

It is at this point that TSGA is again convenient. As we have analysed in Chapter 4, the order in which tasks are allocated is one of the key factors for the performance of this allocation strategy and for this reason, an optimization cycle iterating the order in which the tasks are considered is going to be implemented, following the idea behind TSGA, extensively described in Section 4.2. To this cycle of task order iteration we will, from now on, call *macro allocation cycle* as opposed to the *micro allocation cycle*, defined before. To sum up the differences, and it is important that the reader notes that they are two independent cycles and levels of optimization, *micro allocation cycle* is done to allocate the agents to the best relay points inside one single task, whereas *macro allocation cycle* tests the order in which to allocate the tasks, given a set of tasks. Is is possible, thus, to say that inside one *macro allocation cycle*, many *micro allocation cycle* happen, one for each task in each iteration.

In this allocation cycle, the objective will also be minimizing an objective function that will take into account the total mission time, total distance covered by the drones, as the aforementioned f_1, such that:

$$f_2(\mathbf{P}) = k_3 \times s_{time}(\mathbf{P}) + k_4 \times s_{energy}(\mathbf{P}) \tag{5.2}$$

Note that the coefficients are different from the ones used in the *micro allocation*, as the user has the freedom to tune them differently. For example, if the tasks are not time restrained, the user might set k_3 to zero and take only into consideration the total energy needed. This function will further be modified in order to include some particularities of the relay problem that is being considered.

For now, it is relevant to present an example where the *macro allocation cycle* improves the solution found in comparison to a allocation where the tasks order is not iterated.

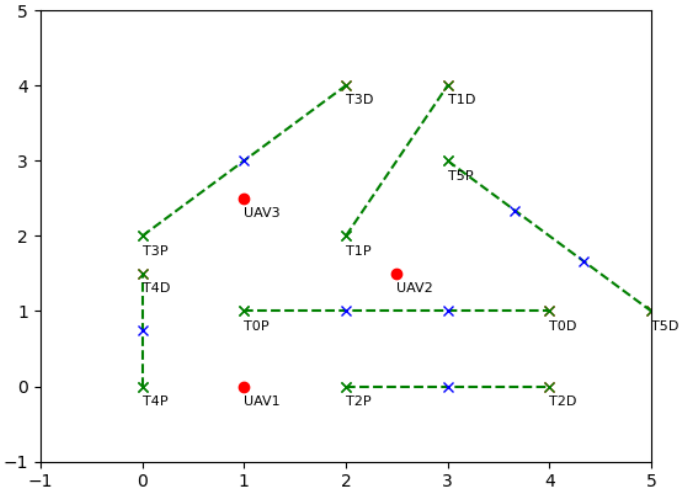

Figure 5.14: Multi task relay development environment

Consider the task environment represented in figure 5.14 where 6 tasks are represented as well as 7 relay points (blue crosses). Two different optimizations will be ran with and without the *macro allocation cycle* in order to study and analyse the improvement that this implementation causes.

Task Allocation Result with no *macro* cycle

Agent	TA
1	[0.0; 1.0; 3.0; 4.1; 5.1]
2	[0.2; 2.1; 4.0; 5.2]
3	[0.1; 2.0; 3.1; 5.0]

Mission time: 937.63
Total distance: 45.06

Table 5.5: Relay Task Allocation 5

Task Allocation Result with *macro* cycle

Agent	TA
1	[4.0; 0.0;1.0; 5.0]
2	[3.1; 0.2; 2.1; 5.2]
3	[3.0; 4.1: 0.1; 2.0; 5.1]

Mission time: 761.73
Total distance: 37.66

Table 5.6: Relay Task Allocation 6

In the first table (Table 5.5) one is able to observe the task allocation result without the concept of TSGA, i.e, the iteration of the order in which tasks are allocated. This can be observed in the schedule of each agent where all tasks appear in numerical order. The second table (Table 5.6) represents the allocation considering both time and distance, with the *macro allocation cycle* and it is possible to observe that the results are better. Remember that the user might tune the k factors in different ways, not only combining zero and one values. These optimizations were performed with the *micro allocation* factors of $k_1 = 1$ and $k_2 = 1$ and with the *macro allocation* factors of $k_3 = 1$ and $k_4 = 1$. It is also possible to observe in Figures 5.15 and 5.16 the task completion times of each of the relay tasks considered.

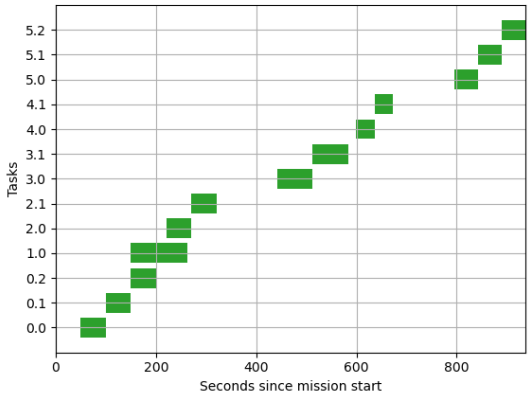

Figure 5.15: Temporal plot without *macro cycle*

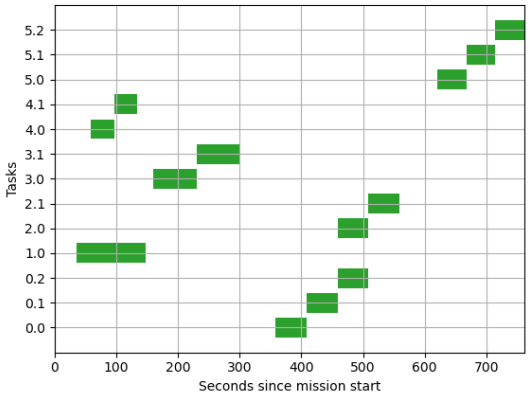

Figure 5.16: Temporal plot with *macro cycle*

Algorithm 6 Macro Allocation Cycle

In: *Tasks, Relay Points, Agents*
Out: *TA*

1: **procedure** MACRO ALLOCATION CYCLE($Task, RelayPoints, Agents$)
2: Define all possible task orders ▷ Similar to TSGA
3: **while** *untested_order* **do**
4: **while** unallocated_tasks **do**
5: Perform *micro allocation cycle* to next task
6: Allocate relay tasks to *Ordered_Coalition*
7: Actualize *unallocated_tasks*
8: Actualize agents info
9: Compute $f_2(P)$
10: **if** *f(P)* $<$ *best* **do**
11: Actualize *best*
12: Actualize TA
13: **return** *TA*

After observing these results, the pseudo-code in Algorithm 6, and after extensive testing on the software was done in order to find coding bugs or logic inconsistencies, the last optimization cycle regarding the relay points was considered. Recalling Figure 5.8, the allocation algorithm is considered implemented at this point and the binary optimization will now be implemented. It is worth mentioning that this binary optimization of the relay points is performed over a finite set of possible relay points, predefined by the user, as described in 5.2. The intention of this optimization phase is, given a set of specific locations where relay maneuvers can happen, find the ones that optimize mission time and energy. For a complete analysis of an environment and to find the optimal locations of the relay points in the overall geography of the mission, it is necessary to iterate the set of possible relay points as it will be done in Section 5.6.

The chosen algorithm to perform the binary optimization is a genetic algorithm given that it is not only an interesting and understandable concept, based on nature, but also it is an algorithm that is well studied and has numerous optimization parameters that the user can change in order to perform relevant studies. This algorithm also allows for modifications to multi-objective optimization which might be a good study in the context of the project this thesis is included in. Moreover, there are many well known and tested Python libraries, modules and solvers to include in this algorithm, and is not necessary to develop the

coding from scratch. A brief explanation of this algorithm can be found in Section 3.2.3.

The genetic solver that was utilized was the module available in [45], developed by Solgi and released in May 2020. It was chosen because of its extremely easiness in the integration and as it also possible to understand the code behind the solver and retrieve certain useful data from it. More documentation and examples can be found in the same reference. In the matter of our project it is important to understand two different fields: how to work with the algorithm, this is, the internal parameters that it provides the ability to change and how to integrate the solver in developed program.

In the next table it is possible to observe the parameters of the solver and a brief explanation to all of them:

GA Solver Parameters

Parameter	Explanation
max_num_iteration	The maximum number of iterations, used as stopping criteria
population_size	Number of chromosomes in each population
mutation_probability	Encodes the probability of a mutation appearing
elit_ratio	The percentage of elite in a population
crossover_probability	The probability of a chromosome to be chosen to perform crossover
parents_portion	Percentage of the next generation automatically filled by parents
crossover_type	The type of crossover to perform (one point, uniform for example)
iteration_without_improvement	Another stopping criteria of non improvement throughout iterations

Table 5.7: Genetic Algorithm Solver Parameters

Regarding the integration of the solver in the algorithm and code developed before, the model receives only one function f (that must return a numeric score), the dimension d of the chromosome (number of decision variables), the algorithm parameters and the variable type (for example real, integer or boolean). Internally, the algorithm iterates a vector X, composed by d variables of the indicated type, that must be an entry of the function f. And that is all that matter concerning integration, proving its easiness and simplicity.

Correspondingly, to adapt our algorithm to implement this solver, an internal function that toggles the relay points from active to inactive based on a received vector (that will in the future be vector X) was developed. The vector is decoded in a way that each boolean variable in it represents that a relay point is active (if 1) and not active (if 0). The vector is ordered firstly by the task numerical order and inside the task by numerical order of the relay points, meaning that, for the example of image 5.14, the generic vector will be (where RP stands for Relay Point):

$$decision\_vector = (RP_{0.1}; RP_{0.2}; RP_{2.1}; RP_{3.1}; RP_{4.1}; RP_{5.1}; RP_{5.2})$$

As an example, take the following two images where two vectors are decoded into the geometry of the task environment based on the map of Figure 5.14 :

Regarding the first figure, only the first and the last relay points are active, according to the vector composition whereas in the second figure, the third, forth and seventh points are active as one may observe in the graphical representations.

Note that there is one variable for each relay point and not for each segment of the task because one

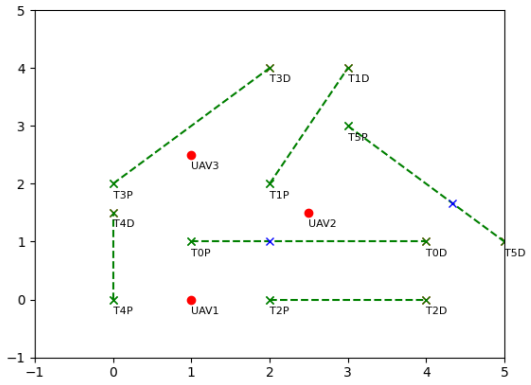

Figure 5.17: Map with $X = (1, 0, 0, 0, 0, 0, 1)$

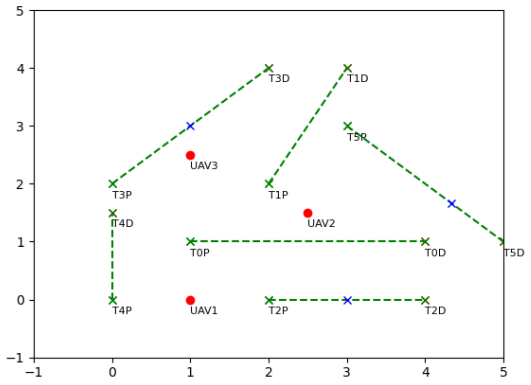

Figure 5.18: Map with $X = (0, 0, 1, 1, 0, 0, 1)$

can not turn inactive the pick up point of the task, without which the task does not exist. Again, the goal is to iterate the relay points and find an optimal set, regarding the possible locations.

In fact, the strategy of this implementation is that the solver iterates the X vector and finds the combination of active relay points that is optimal regarding the objective function. The f function is simply the algorithm that combines the *macro allocation cycle* and the *micro allocation cycle*, that means, the modified TSGA with relays, returning the score of the f_2 objective function to the genetic operation. This achievable because the macro and micro allocation cycles are constructed in a generic way such that any combination of task set and relay point set might be an input to the algorithm. By this and by the means of the decoding function that receives the vector and retrieves the active relay points, it is possible to implement the genetic optimization using these as tools for the relay points optimization. In Figure 5.19 and in Algorithm 7 it is possible to observe a chart showing the information flow, or the code architecture behind this last optimization cycle and the respective pseudo-code.

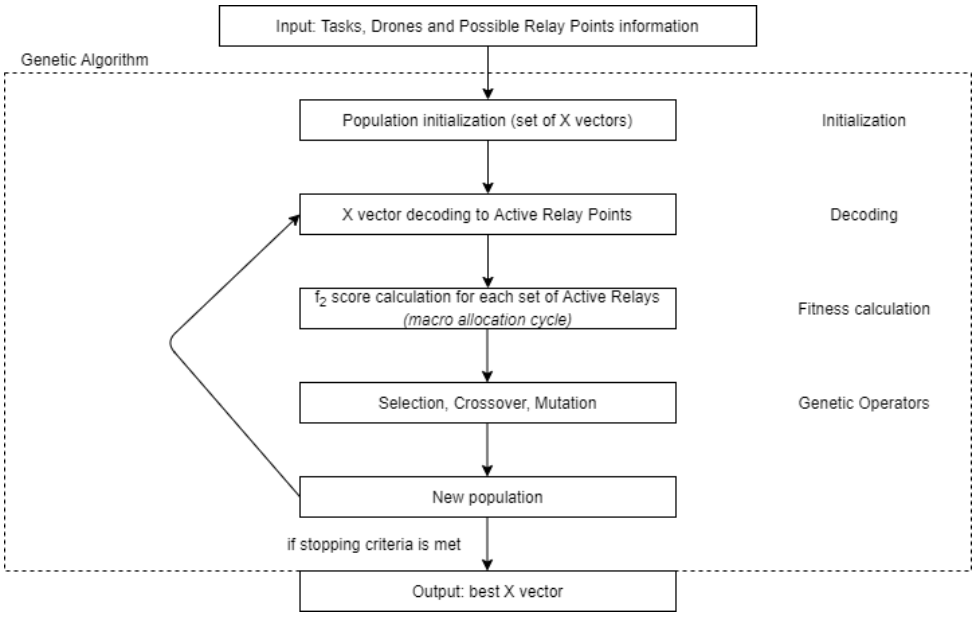

Figure 5.19: Architecture of TSGA with Relay Points Genetic Optimization

Algorithm 7 Implemented Algorithm with Relays

In: *Algorithm Parameters, Objective Function, Tasks, Drones, Possible Relay Points*
Out: *Best Set of Active Relay Points*

```
1: procedure COMPLETE IMPLEMENTED ALGORITHM
2:     Population initialization (Set of X vectors)
3:     while Stop Criteria not met do
4:         while X vectors do
5:             Decode X vector into the active set of relay points
6:             Macro Allocation Cycle (Evaluation of the fitness f_2 of X vector)
7:         Selection
8:         Crossover
9:         Mutation
10:        Active population actualization (New set of X vectors)
11:    return Best X vector
```

After the implementation and complete integration of the genetic algorithm solver, an optimization procedure will be performed with k_1,k_2,k_3 and $k_4 = 1$, meaning that in all levels, both time and energy are being considered and with the default solver parameters for the genetic procedures.

Before running, a quick thinking about what to expect as an optimal solution led to the conclusion that the optimal solution would be to have no relay points. This is because as there is no advantage in what concerns time because the fleet of drones in this environment is homogeneous, this is, all drones have the same velocity and it is the equal for the time mission if a relay maneuver happens or not, and in what concerns battery it is always better not to have relay maneuvers because it saves all the distance that a drone has to cover from where it is to the relay point, whereas without relays, the task is continuous and performed by just one drone.

After performing the optimization procedure, the optimal solution (X vector) that the algorithm retrieved is $X = (0, 0, 0, 0, 0, 0, 0)$, meaning that there are no relay points that enhance the mission performance, as expected, corresponding to the task allocation in Table 5.8:

Task Allocation Result - Optimal Solution

Agent	TA
1	[0.0,2.0]
2	[1.0; 5.0]
3	[4.0; 3.0]

Mission time: 411.81
Total distance: 22.44

Table 5.8: Relay Task Allocation 7

Moreover, in Figure 5.20 it is represented the convergence of the algorithm, i.e, the best score in each one of the iteration such that on can analyse the evolution of the algorithm, for 10 different runs, given the stochastic behaviour of the algorithm.

To this extension, the algorithm is not very interesting as we were able to predict the result before performing the optimization, and if that happens, an optimization procedure is only needed for validation. However, there are differences and particularities of the problem that can be thought, were this algorithm can be validated and can also has a very interesting application. The first one (and more straight forward) is when the drone battery has a limit and the relay maneuvers are called as a necessity of the drones,

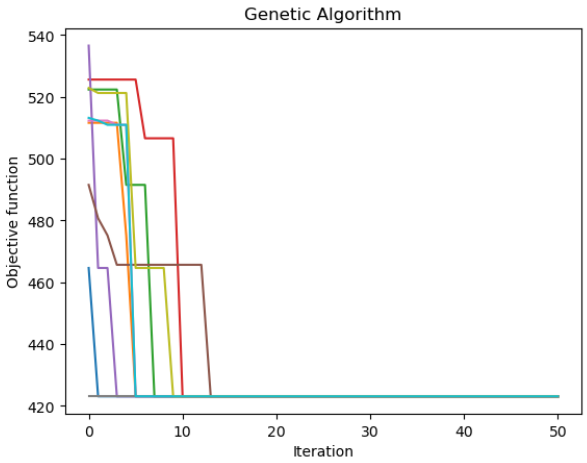

Figure 5.20: Algorithm convergence plot

meaning that the appearance of relay points will provide the possibility of some tasks to be performed. Imagine the simple case of the agents having an energy limit of 10 energy units (and a consumption of 1 energy unit per distance unit) and existing a task with more than 10 distance units length. Only the appearance of a relay point can enable the task to be performed. After this, another interesting analysis would be implementing the recharge bays as we analysed in Chapter 4 and also understand the behaviour of the system.

At this point, and without implementing any additional algorithm capabilities an interesting analysis and a good validation to the implementation of the genetic solver can be imposing a minimum number of relay points in the solution. This is done by verifying if the sum of the X vector is higher than the number of minimum relay points the user wants to consider, and if not, attributing a big score, meaning that the algorithm will not opt neither converge to these solutions.

As an example let us analyse what happens if a minimum number of 3 relay points is established. After performing the optimization, the best solution retrieved was $X = (1, 1, 0, 0, 0, 0, 1)$, representing the map represented in Figure 5.21 with the time distribution represented in Figure 5.23. Given the stochastic behaviour of the algorithm, 8 simulations where performed and 2 of them retrieved different results from this. Despite this fact, and after comparison of the respective objective function values, we can conclude that this X is the best one (please see Table 5.9). The 2 simulations that did not achieve this result confirm that one must be extremely careful when writing conclusions when working with stochastic procedures and also calls for attention in the set up of the model where the genetic parameters might have deep influence on the convergence and quality of the results. These optimizations where performed with a population of size 20 and imposing a number of iterations without improvement of 15. All the other parameters are the default ones from the solver. A plot of the convergence of one of the simulations is also presented. After this, a quantitative comparison of the obtained optimal result, the other 2 retrieved results and some others random possible combinations with the respective score function values is presented with the intent of validating the obtained result, noting that there is no other X vector that obtains a better result in the objective function than the one that is the optimal result of the study.

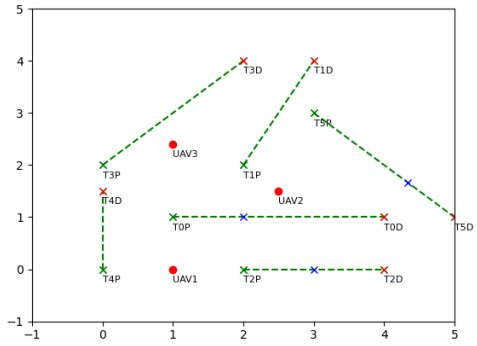

Figure 5.21: Map with $X = (1, 1, 0, 0, 0, 0, 1)$

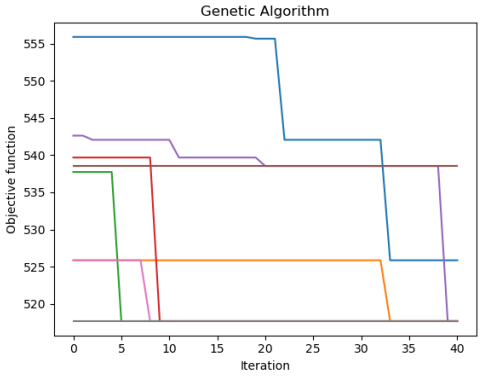

Figure 5.22: Convergence analysis

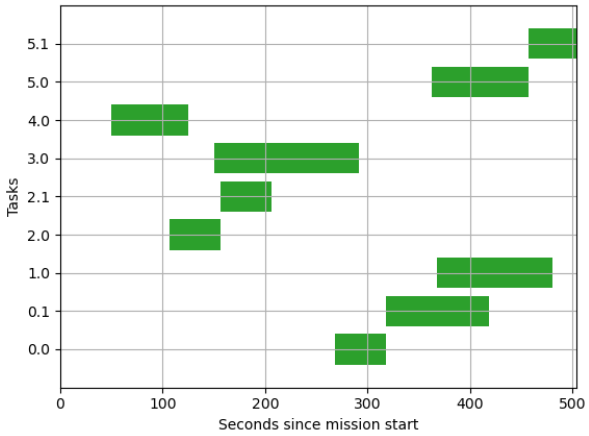

Figure 5.23: Time plot of the optimized relay tasks

Results comparison

X vector	Mission time	Total distance	Score function
(1,1,0,0,0,0,1)	543.45	26.90	556.90
(1,1,1,0,0,0,0)	523.93	27.59	537.73
(0,0,0,1,1,0,1)	541.80	28.01	555.81
(1,0,1,0,1,0,0)	525.98	27.39	539.68
(0,0,0,1,1,1,0)	541.81	28.19	555.90
(0,0,1,0,0,1,1)	563.46	27.23	577.08

Table 5.9: Results comparison for different sets of Active Relay Points

5.5 Algorithm Characterization

In this section, similarly to the last chapter, we will perform some analysis on the algorithm complexity increase when some dimension on the input variables are increased. Comparing with the analysis in Section 4.5, the study of the number of bays will not be performed, given the reason that no recharge

options are implemented in this environment, and the study of the average number of agents per task does not apply as well, given the assumption that for this implementation, tasks should only be carried by one drone. It is again worth noting that the simulations were ran in a computer with an Intel(R) Core(TM) i7-770HQ CPU @ 2.80GHz processor, with an installed RAM of 16GB. This is not expected to influence the variation of the complexity with the different dimension of the inputs but if one would want to analyse the absolute value of the run time present in the images, it is important in order to compare with other works.

Given this, we will study the increase of complexity in two different levels. The ”upper level” where all the algorithm is being ran including the genetic binary optimization, in which the parameters of population size and number of iterations will be analysed, and a ”lower level” where the influence of some parameters will be studied only in the *macro allocation cycle* that is mostly the TSGA algorithm of Chapter 4. At this level, the influence of the number of tasks is assumed to be the same as the influence reported in Section 4.5 once the combinatorial nature of the allocation is preserved. For this reason we will study the influence of the total number of agents and of the total number of active relay points. The total number of drones is expected to have a light greater influence on the complexity of the problem due to the *micro allocation cycle*, where a combinatorial search is performed on the members of the coalition for each task. The total number of relays points is also expected to lightly influence the run time of the algorithm. The number of relay points per task is expected to have a great influence, because the *micro allocation cycle* also has a combinatorial nature. For this reason, this variable will also be tested. It is worth noting that in the genetic algorithm, one *macro allocation cycle* is performed for each individual in each iteration. For this reason, an increase in complexity in the ”lower level” analysis has an amplification factor given approximately by the product of the population size and the number of iterations, when talking about the integral algorithm. All simulations were performed with the k factors of 1, in consideration both time and energy in the all the optimization levels.

We will start with the ”lower level” complexity analysis. Each point on the plots below was retrieved, similarly to the approach on Section 4.5, from 5 different environments, in order to eliminate influence of the map spacial characteristics on this analysis and each of those environments will be run 10 times in order to eliminate some computational occupation of the computer. All the environments will have 6 tasks. For the study of the influence of the total number of relays, the number of drones is fixed in 3 and the average number of relays per task is 1. For the analysis regarding the number of drones, the average number or relays per tasks is one and the total number of relays is 6. Finally, to the study of the influence of the number of relays per task, 5 agents will be initialized (in order to be able of studying until 5 relays per task) and 6 tasks will be maintained. For the study of the influence of the number of iterations, the population size is kept constant at 8 individuals and regarding the analysis of the influence of the population size, the number of iterations is as well constant and equal to 20 iterations.

Considering Figures 5.24 and 5.25 we might observe that the influence of the number of available drones is not high, given that the run time is almost stable in the final 3 points of the plot, and that the number of total relays tend to have an approximately linear influence of the complexity of the algorithm although the correlation with this variable is extremely low as we might conclude from the error bars in Figure 5.25.

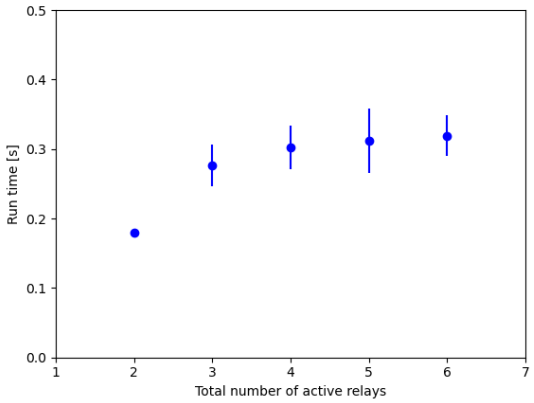

Figure 5.24: Influence of the number of drones

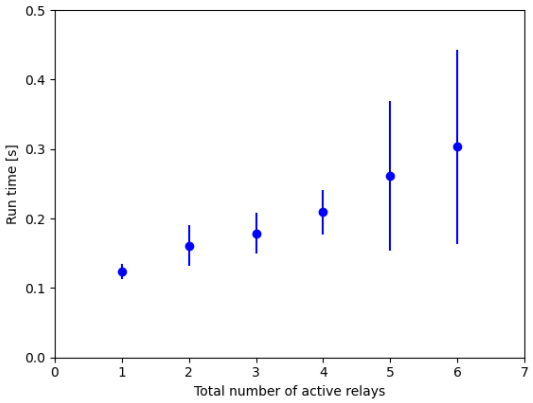

Figure 5.25: Influence of the number of total relays

This suggest that as we increase the number of relay tasks, the complexity of the algorithm also tends to increase but also that this factor has no influence by itself, suggesting that the number of total relay tasks might be a consequence of other more correlated variable. As we have stated before, this variable is expected to be the average number of relay points per task, given the combinatorial allocation inside the *micro allocation cycle*. The plot depicted in the next image, proves that this assumption was correct.

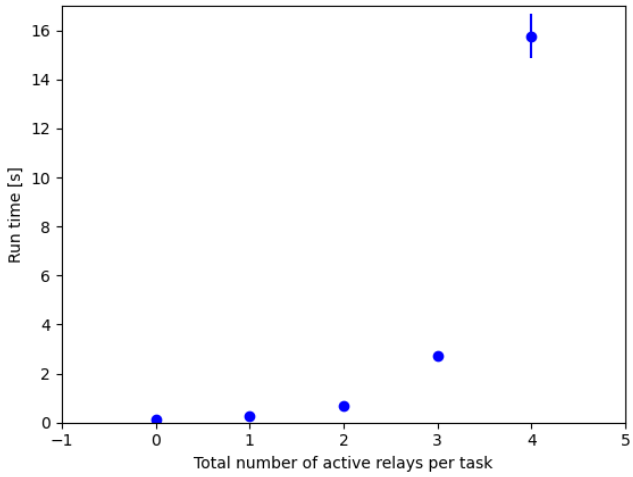

Figure 5.26: Influence of the number of relays per task

In fact, Figure 5.26 shows a combinatorial influence of the average number of relays per task.
The results for the influence of the genetic parameters of population size and number of iterations will be presented next. For this simulations, 3 runs in the same environment will be performed. Differences on the environment are considered internally in the genetic algorithm, as the X vector will be iterated. Also the influence of the RAM memory temporary occupation can be neglected given the magnitude of the run times. For comparison, it is important noting that the 3 runs (on a environment with 3 drones, 6 tasks and 1 relay per task) are performed to account for the stochastic behaviour of the algorithm.
Both variables seem to have a linear influence (with different slopes) on the run time as it was expected, once, as stated before, these modifications decide how many times the *major allocation cycle* is being

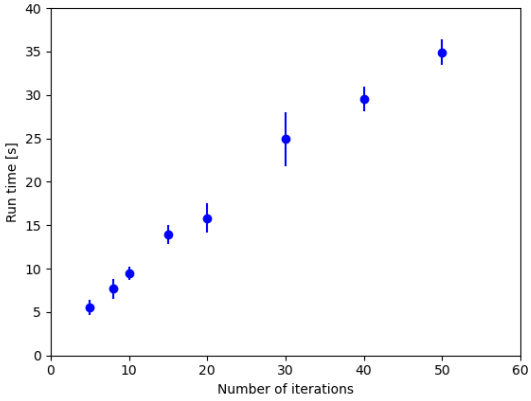

Figure 5.27: Influence of the number of iterations

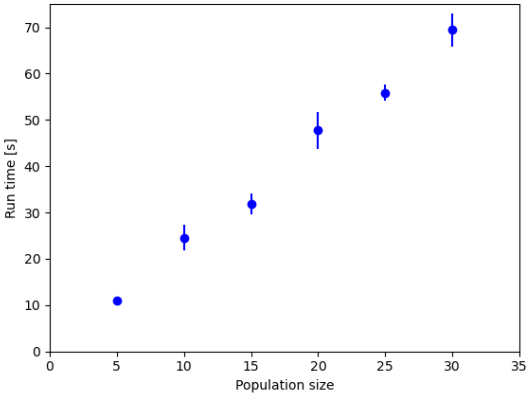

Figure 5.28: Influence of the population size

performed.

5.6 Case Study Simulation Results

In this section, a direct application of the developed algorithm with relay maneuvers is presented. It is worth remembering that the goal of this type of study is not only retrieving the optimal task allocation result, but also in this chapter to understand the optimal locations for the relay maneuvers to happen. As we have seen before in Section 5.4, battery limitation and recharge possibility are not implemented in this environment and for this reason, a study of a homogeneous fleet (where all drones have the same characteristics) will always result in a environment with no relay points, since there is no incentive for the relay maneuvers to happen. Despite this fact, this algorithm as it is can be used to study the behaviour of heterogeneous fleets and the advantages of introducing one or more drones with a higher velocity than the others. This is where this case study will focus. For this reason, this environment can be thought to be an environment where the drones have enough battery to perform all tasks, which with the advances in battery technology and particularly battery endurance, might be plausible in some circumstances.

The general line of thought of our analysis will be first to define the task environment (with no relay points). We will initiate the environment with long tasks in order to be able to split the tasks with possible locations to the relay points that are not too close to each other. It is also worth remembering that the maximum number of sections in a tasks can be split is the number of available agents in the simulation environment. After this, perform one allocation to a homogeneous fleet (witch will retrieve an empty optimal relay points set), in order to have a comparison baseline in what concerns mission total time. Following, we will decide the heterogeneous fleets (the drone configurations) that we want to analyse, and perform the optimization procedure to find the optimal relay points. After this, the complete algorithm with the binary genetic optimization will be ran for different possible relay points. For instance, a first optimization will be performed with one possible relay point in the middle of the task, after with 2 relay points in the thirds of the task followed by one with 3 relay points in the quarters of the task and successively until we reach the limit of the number of agents. In the limit, the tasks will have so many

possible relay points that the optimization will almost be a continuous optimization along the task length. This is important to understand if the successive divisions still influence the result of the mission time, otherwise one is not able to state that those are the optimal locations of the relay points. Also, before the optimization procedure, an analysis on the influence of the genetic parameters of the optimization need to be performed, in order to observe if the algorithm is converged, meaning that the parameters do not influence the achieved result and that the algorithm is able to reach a solution.

After this brief explanation, let us present the environment of this case study, composed by 6 different tasks and 5 agents, and represented in Figure 5.29. We will assume that all agents begin the mission in origin of our coordinates system, the point $x = 0$ and $y = 0$.

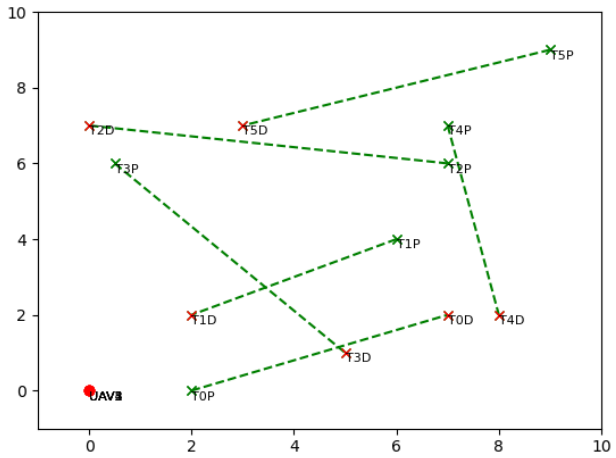

Figure 5.29: Case Study 2 environment

Let us also define 3 classes of drones and respective velocity and consumption:

- Drone 1: velocity of 0.02 distance units per time unit and consumption of 1 energy unit per distance covered
- Drone 2: velocity of 0.06 distance units per time unit and consumption of 1.2 energy unit per distance covered
- Drone 3: velocity of 0.08 distance units per time unit and consumption of 1.5 energy unit per distance covered

The optimizations were conducted giving more importance to mission time compared with consumed energy. The baseline result for comparison, with a fleet of 5 drones of type 1 is as follows in Table 5.10 and Figure 5.30.

Given this, let us imagine that one has the money to upgrade 2 drones to drones of type 2 (option 1) or instead upgrade 2 drones to type 3 (option 2), that have an higher velocity but also an higher consumption. For this decision, the optimal relay points will be discovered (or at least the most optimal considering the number of drones limitation), and the total mission time and energy consumed will be compared.

Task Allocation Result

Agent	TA
1	[0.0;2.0]
2	[1.0]
3	[3.0]
4	[4.0]
5	[5.0]

Mission time: 952.62
Total energy: 76.94

Table 5.10: Relay Task Allocation 8

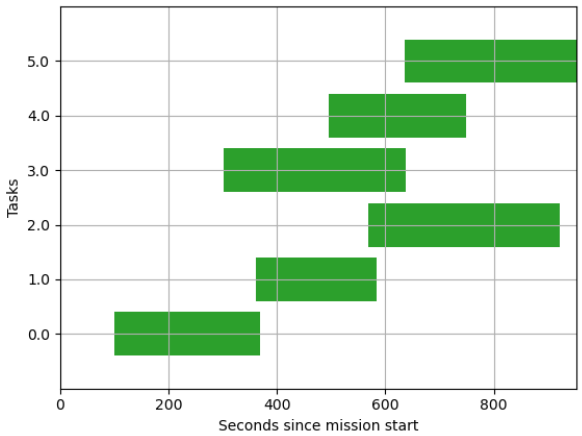

Figure 5.30: Case study time plot

For the study of the influence of the genetic parameters, to choose the parameters for which the algorithm is converged, we will perform the analysis for a 10 individuals population size and observe the sufficient number of iterations to assume a converged algorithm. We will perform the analysis for the option 1 with 2 relay points in each task and choose the parameters converged enough to assume that the other runs are also converged, as we will see next. For each iteration number, the algorithm was ran 3 different times such that the analysis is not biased by stochastic behaviour.

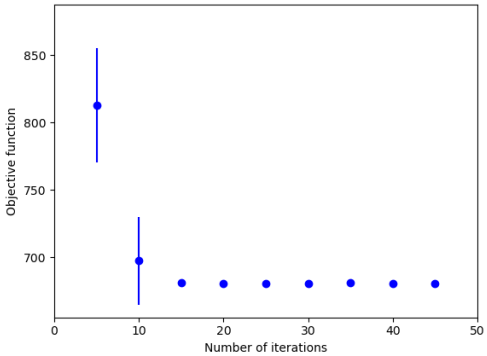

Figure 5.31: Genetic Convergence Case Study 2

Given the analysis in Figure 5.31, we consider that for 35 iterations the algorithm is converged and the

analysis from now on will be held with a population of 10 individuals and 35 iterations. The procedure from now on will be split the tasks in successively more relay points, beginning in 1 until 5. The best objective functions of all the runs will be the best solution that the algorithm was able to reach. In Figures 5.32 and 5.33 the environment with 2 and 3 relays points in each task.

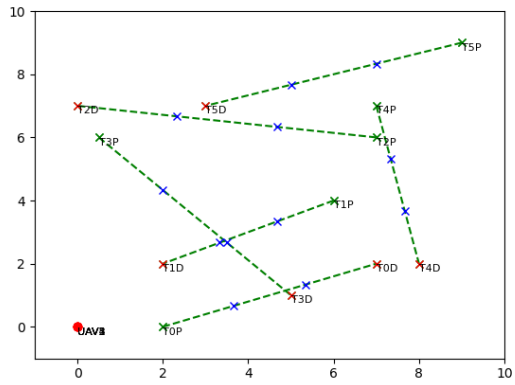

Figure 5.32: 2 relay points per task

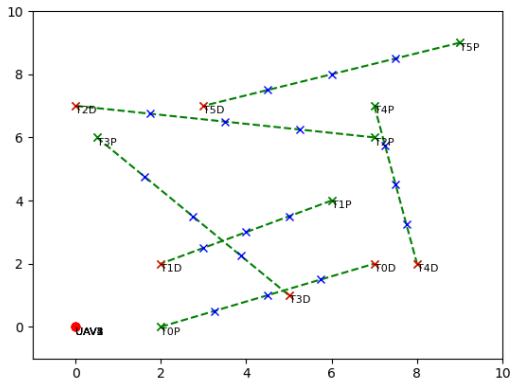

Figure 5.33: 3 relay points per task

Figures 5.34 and 5.35 represent the value of the objective function for each number of divisions of the tasks. Remembering that this is a minimization problem, the solution of the problem for each of the options is considered to be the point of minimum objective function value.

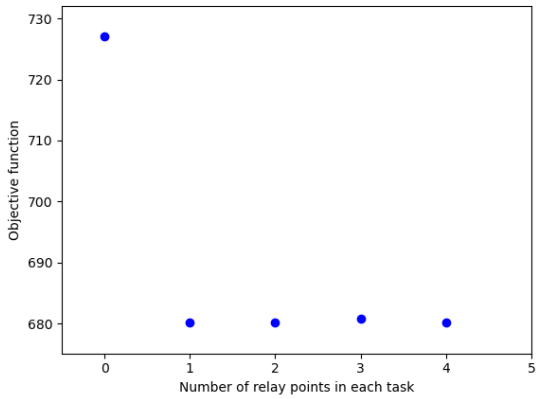

Figure 5.34: Option 1

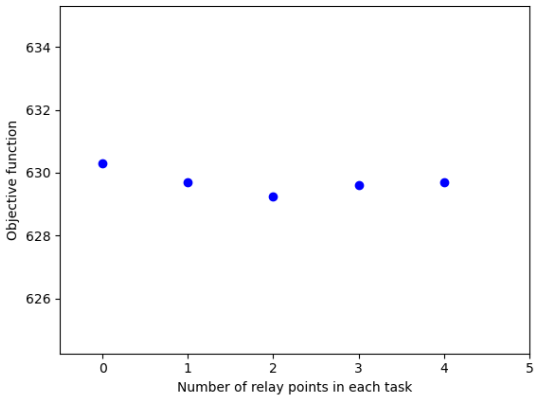

Figure 5.35: Option 2

From Figure 5.34 we see that the best result was obtained with 2 possible relay points in each task for option 1 and from 5.35 we can observe that the best result was obtained also with 2 relay points in each task for option 2. It is also curious to observe that with the fleet of option 1 the introduction of relay points on the environment represented a great enhancement of the result (difference from 0 relays per task to 1 relay per task), whereas with the fleet of the second option this improvement was smaller.

Tables 5.11 and 5.12 represent on the left, the task allocation for option 1 and on the right the task allocation for option 2. Please note that the upgraded drones are number 4 and 5. Figures 5.36 and 5.37 represent the final maps with the selected relay points, and Figures 5.38 and 5.39 represent the

time plot for the achieved task allocations.

Task Allocation Result Option 1

Agent	TA
1	[0.0]
2	[1.1]
3	[1.2]
4	[3.0;1.0;2.0]
5	[5.0;4.0]

Mission time: 488.77
Total energy: 87.77

Table 5.11: Relay Task Allocation 9

Task Allocation Result Option 2

Agent	TA
1	[0.0]
2	[0.1]
3	[3.0]
4	[4.0;1.0;3.1]
5	[5.0,2.0]

Mission time: 385.12
Total energy: 102.48

Table 5.12: Relay Task Allocation 10

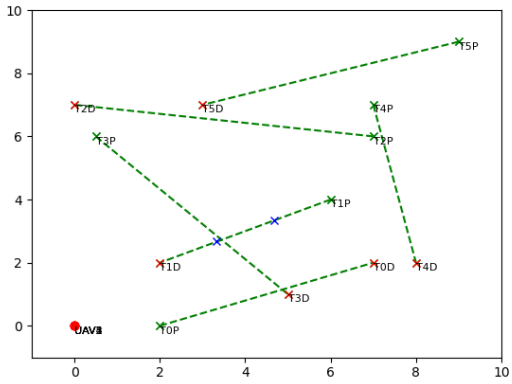

Figure 5.36: Option 1 map

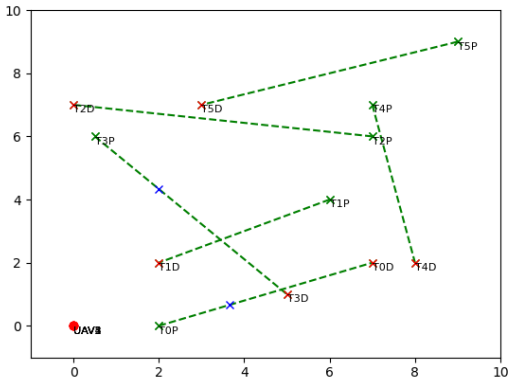

Figure 5.37: Option 2 map

We can observe that both options resulted in a decrease of mission time and and increase of mission energy. This was expected since we increased the fleet average velocity and increased the fleet average consumption. Moreover, one might observe that the two upgraded drones are the ones that perform more tasks, which makes sense given that are the two quickest agents. In order to minimize time, the system tends to give more tasks to both these drones. In overall result, option 2 is considered to be a better option, as far as the objective function is concerned. However, one might also observe that in what concerns energy total consumption, option 1 performs better that option 2.

To conclude, it is important to register the genetic parameters with which these analysis were performed in order to guarantee that other researchers can replicate the obtained results and compare other achieved conclusions. These parameters can be observed in Table 5.13.

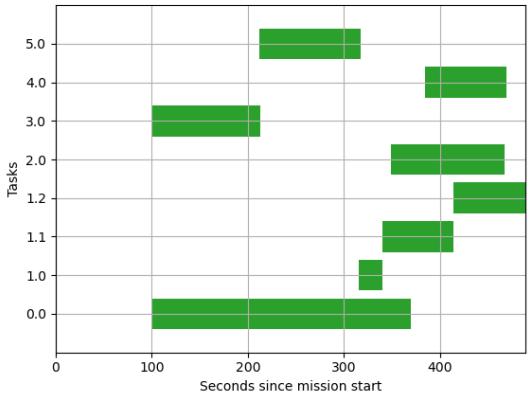

Figure 5.38: Time plot option 1

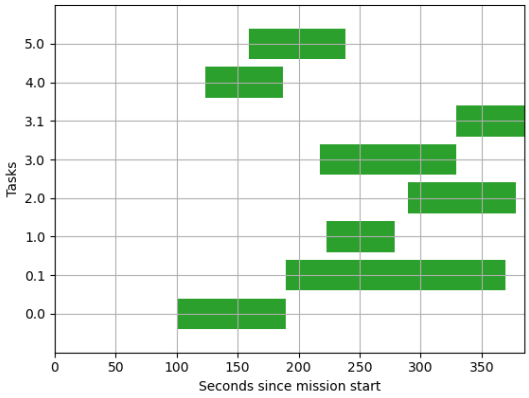

Figure 5.39: Time plot option 2

GA Solver Parameters - Case Study

Parameter	Value
max_num_iteration	35
population_size	10
mutation_probability	0.01
elit_ratio	0.01
crossover_probability	0.5
parents_portion	0.2
crossover_type	Uniform
iteration_without_improvement	None

Table 5.13: Case Study Solver Parameters

Chapter 6

Conclusions

6.1 Contributions

The work developed in the context of this thesis combines multi-agent systems study applied with modern urban logistics, given that it aimed to study a multi-drone cooperative parcel delivery system.

The main objectives were accomplished since an already known algorithm, Task Sequential Greedy Algorithm, was implemented and modified to have not only recharge possibilities but also the relay maneuvers. At first we tackled the problem of cooperative task allocation in the sense that more that one drone were needed to perform each one of the task. After, in Chapter 5, the cooperation relies on the coordination and time synchronization to assure the relay maneuvers.

This work also had the objective of producing a generic algorithm, meaning that the algorithm and the framework are coded in a way that any type or size of the data sets can be considered as inputs. In practice, any fleet of drone (both homogeneous and heterogeneous) and any type of tasks (with diverse weights or lengths) can be studied in this algorithm.

Both stages of the algorithm (with and without relay maneuvers) were characterized from a evaluative point of view, meaning that the user has the information about the performance of the algorithm in certain conditions and also its limitations. Besides this, two examples of practical application of the developed algorithms were presented, in order to better explain the usage and utility of the study that was conducted.

In what concerns a broader analysis of the achieved results, one is able to state that this modified algorithm achieves a feasible solution for any studied environment. It is also worth noting again, that this works is expected mainly to provide an initial framework for the inclusion of recharge procedures and relay maneuvers without severe optimality and performance objectives (although always trying to ensure the best possible solution). Given this, all the approaches and implementation decisions are explained along this thesis and were thought to guarantee the maximum quality of the achieved solutions. However it is not possible to clearly evaluate the quality of the solutions once there is no available benchmark data set for cooperative task allocation for comparison and also because the theoretical optimal solution for each environment is not know (and in general difficult to achieve for real-world scenarios) given

the complexity of the problem. For this reason, the achieved results are also expected to create a comparison baseline for further study regarding this subject.

6.2 Future Work

Regarding future work there are several ways to enrich this algorithm, to deepen the analysis or even to enhance the correlation with the reality of the simulations:
Firstly, in the implementation of TSGA, time windows can be implemented, such as in the baseline article. This modification aims to provide the algorithm the possibility of evaluate the task according to their priority in time. Another good implementation for the algorithm would be time varying velocity. This means that with this implementation, the drone should understand if he is the limiting agent of a task and if not, it would make the decision to slow down or just hover in order to save energy. Another modification that would provide a more realistic simulation environment would be to consider a 3D space, with ascent and descent and hovering of the drones and the time and energy associated with it.
Moreover, an introduction of a real drone model with uncertainty not only in sensors information but also in its flight performance and attitude could lower the gap between the created environment and real world simulations.
One of the biggest limitations of the algorithm we implemented and modified is its combinatorial nature. As a consequence, when bigger data sets are considered, the run time of the algorithm is too large to consider using it in real life systems. Data clustering techniques can be implemented in order to tackle this problem.
Also, an interesting approach would be given by a multi-objective optimization procedure, given the mission time, energy, operational time of relays or recharges. The user would clearly see the trade offs between those variables and their impact on the objective function.
To conclude, the study of a decentralized approach to this algorithm would also be of interest, not only to study the differences regarding the system response but also because this modification would allow for the usage of bigger data sets without escalating in computational effort or run time, thus improving the scalability of the strategy.

Bibliography

[1] Replace research project web page. Available at `http://replace.isr.tecnico.ulisboa.pt/` (2020/10/27).

[2] Robin Kellermann, Tobias Biehle, and Liliann Fischer. Drones for parcel and passenger transportation: A literature review. *Transportation Research Interdisciplinary Perspectives*, 4, 2020.

[3] Mireia Roca-Riu and Monica Menendez. Logistic deliveries with drones: State of the art of practice and research. In *19th Swiss Transport Research Conference (STRC 2019)*. STRC, 2019.

[4] Sung Hoon Chung, Bhawesh Sah, and Jinkun Lee. Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research*, page 105004, 2020.

[5] Corey Schumacher. AIAA 2003-5664 UAV Task Assignment with Timing Constraints. *Transition*, (August), 2003.

[6] Phillip R. Chandler, Meir Pachter, Steven R. Rasmussen, and Corey Schumacher. Multiple task assignment for a UAV team. *AIAA Guidance, Navigation, and Control Conference and Exhibit*, (August):1–10, 2002.

[7] Jia-lei Liu, Zhi-guang Shi, and Yan Zhang. A New Method of UAVs Multi-target Task Assignment. *DEStech Transactions on Engineering and Technology Research*, (icmeit):388–394, 2018.

[8] Gyeongtaek Oh, Youdan Kim, Jaemyung Ahn, and Han Lim Choi. PSO-based Optimal Task Allocation for Cooperative Timing Missions. *IFAC-PapersOnLine*, 49(17):314–319, 2016.

[9] P B Sujit J M George and Randy Beard. Multiple UAV Task Allocation using Particle Swarm Optimization. (August), 2008.

[10] Yohanes Khosiawan and Izabela Nielsen. Indoor UAV scheduling with Restful Task Assignment Algorithm. pages 1–29, 2017.

[11] Gyeongtaek Oh, Youdan Kim, Jaemyung Ahn, and Han-Lim Choi. Task Allocation of Multiple UAVs for Cooperative Parcel Delivery. In *Advances in Aerospace Guidance, Navigation and Control*. 2018.

[12] Nuri Ozalp, Ugur Ayan, and Erhan Oztop. Cooperative multi-task assignment for heterogonous UAVs. *Proceedings of the 17th International Conference on Advanced Robotics, ICAR 2015*, (July):599–604, 2015.

[13] Lu Zhen. Task assignment under uncertainty: Stochastic programming and robust optimisation approaches. *International Journal of Production Research*, 53(5):1487–1502, 2015.

[14] Mehdi Alighanbari and Jonathan P. How. A robust approach to the UAV task assignment problem. *International Journal of Robust and Nonlinear Control*, 18(2 SPEC. ISS.):118–134, 2008.

[15] Jesus Capitan, Matthijs T J Spaan, Luis Merino, and Anibal Ollero. Decentralized Multi-Robot Cooperation with Auctioned POMDPs. pages 3323–3328, 2012.

[16] Hosam Hanna. Decentralized Approach for Multi-Robot Task Allocation Problem with Uncertain Task Execution.

[17] Andrew K. Whitten, Han Lim Choi, Luke B. Johnson, and Jonathan P. How. Decentralized task allocation with coupled constraints in complex missions. *Proceedings of the American Control Conference*, pages 1642–1649, 2011.

[18] Trevor Campbell, Luke Johnson, and Jonathan P How. Multiagent Allocation of Markov Decision Process Tasks. *2013 American Control Conference*, pages 2356–2361, 2013.

[19] Malcolm Strens and Neil Windelinckx. Combining Planning with Reinforcement Learning for Multi-robot Task Allocation. pages 260–274, 2005.

[20] Baoxiang Li, Dmitry Krushinsky, Hajo A Reijers, and Tom Van Woensel. The share-a-ride problem: People and parcels sharing taxis. *European Journal of Operational Research*, 238(1):31–40, 2014.

[21] Ngoc-Quang Nguyen, Nguyen-Viet-Dung Nghiem, Phan-Thuan Do, Khac-Tuan LE, Minh-Son Nguyen, and Naoto Mukai. People and parcels sharing a taxi for tokyo city. In *Proceedings of the Sixth International Symposium on Information and Communication Technology*, pages 90–97, 2015.

[22] Veaceslav Ghilas, Emrah Demir, and Tom Van Woensel. Integrating passenger and freight transportation: Model formulation and insights. *Proceedings of the 2013 Beta Working Papers; Technische Universiteit Eindhoven: Eindhoven, The Netherlands*, 441:1–23, 2013.

[23] Renaud Masson, Anna Trentini, Fabien Lehuédé, Nicolas Malhéné, Olivier Péton, and Houda Tlahig. Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics*, 6(1):81–109, 2017.

[24] Wenyi Chen, Martijn Mes, and Marco Schutten. Multi-hop driver-parcel matching problem with time windows. *Flexible services and manufacturing journal*, 30(3):517–553, 2018.

[25] Huiting Hong, Xin Li, Daqing He, Yiwei Zhang, and Mingzhong Wang. Crowdsourcing incentives for multi-hop urban parcel delivery network. *IEEE Access*, 7:26268–26277, 2019.

[26] Yuncheol Kang, Seokgi Lee, and Byung Do Chung. Learning-based logistics planning and scheduling for crowdsourced parcel delivery. *Computers & Industrial Engineering*, 132:271–279, 2019.

[27] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of industrial information integration*, 6:1–10, 2017.

[28] Ocident Bongomin, Aregawi Yemane, Brendah Kembabazi, Clement Malanda, Mwewa Chikonkolo Mwape, Nonsikelelo Sheron Mpofu, and Dan Tigalana. The hype and disruptive technologies of industry 4.0 in major industrial sectors: A state of the art. 2020.

[29] Jesús Sánchez-García, JM García-Campos, Mario Arzamendia, D Gutierrez Reina, SL Toral, and D Gregor. A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications. *Computer Communications*, 119:43–65, 2018.

[30] Sameera S Ponda, Luke B Johnson, Andrew N Kopeikin, Han-Lim Choi, and Jonathan P How. Distributed planning strategies to ensure network connectivity for dynamic heterogeneous teams. *IEEE Journal on Selected Areas in Communications*, 30(5):861–869, 2012.

[31] Andrew N Kopeikin, Sameera S Ponda, Luke B Johnson, and Jonathan P How. Dynamic mission planning for communication control in multiple unmanned aircraft teams. *Unmanned Systems*, 1(01):41–58, 2013.

[32] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. Networked drone cameras for sports streaming. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 308–318. IEEE, 2017.

[33] Seon Jin Kim, Gino J Lim, and Jaeyoung Cho. Drone relay stations for supporting wireless communication in military operations. In *International Conference on Applied Human Factors and Ergonomics*, pages 123–130. Springer, 2017.

[34] Jonathan Gross, Jay Yellen, and Ping Zhang. *Handbook of Graph Theory*. CRC Press, 2014.

[35] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition, 2006.

[36] Steve Skiena. *The Algorithm Design Manual*. Telos Pr, New York, 1997.

[37] Ian Chivers and Jane Sleightholme. An introduction to algorithms and the big o notation. In *Introduction to Programming with Fortran*, pages 359–364. Springer, 2015.

[38] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3 edition, 2009.

[39] Frederick Hillier and Gerald Lieberman. *Introduction to Operations Research*. McGrawHill, 8 edition, 2006.

[40] Charles Darwin. *The works of Charles Darwin, Volume 16: The origin of species, 1876*. NYU Press, 2010.

[41] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[42] Yoshida Yukiko and Adachi Nobue. A diploid genetic algorithm for preserving population diversity—pseudo-meiosis ga. In *International Conference on Parallel Problem Solving from Nature*. Springer, 1994.

[43] Ke-Lin Du, MNS Swamy, et al. Search and optimization by metaheuristics. *Techniques and Algorithms Inspired by Nature; Birkhauser: Basel, Switzerland*, 2016.

[44] Jouni Smed and Harri Hakonen. *Algorithms and networking for computer games*. Wiley Online Library, 2006.

[45] Ryan Solgi. Genetic algorithm python module. Available at `https://pypi.org/project/geneticalgorithm/` (2020/09/27).

[46] BB Choudhury and BB Biswal. Alternative methods for multi-robot task allocation. *Journal of Advanced Manufacturing Systems*, 8(02):163–176, 2009.

[47] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[48] Silvia Andrea Suárez Barón. *Dynamic task allocation and coordination in cooperative multi-agent environments*. Universitat de Girona, 2011.

[49] N. Miyata, J. Ota, T. Arai, and H. Asama. Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *IEEE Transactions on Robotics and Automation*, 18(5):769–780, 2002.

[50] M. N. S. Swamy K. Thulasiraman. *Graphs: Theory and Algorithms*. Wiley-Interscience, 1 edition, 1992.

[51] Prof. Gregory Z. Gutin (auth.) Prof. Jørgen Bang-Jensen. *Digraphs: Theory, Algorithms and Applications*. Springer Monographs in Mathematics. Springer-Verlag London, 2009.

[52] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[53] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[54] Maad Mijwil. Genetic algorithm optimization by natural selection. 2016.

[55] Mehdi Alighanbari. Task assignment algorithms for teams of uavs in dynamic environments. Master's thesis, Massachusetts Institute of Technology, 2014.

[56] Marta Marques. Trajectory planning and control for drone replacement during formation flight. Master's thesis, Instituto Superior Técnico, 2018.

[57] Thareswari Nagarajan and Asokan Thondiyath. An algorithm for cooperative task allocation in scalable, constrained multiple robot systems. *Intelligent Service Robotics*, 7(4):221–233, 2014.

[58] Shayegan Omidshafiei. Decentralized teaching and learning in cooperative multiagent systems. Master's thesis, Massachusetts Institute of Technology, 2015.

[59] Marta Marques, Bruno J Guerreiro, Rita Cunha, and Carlos Silvestre. Trajectory planning and control for drone replacement for multidrone cinematography. *IFAC-PapersOnLine*, 52(12):334–339, 2019.

[60] Lavinia Amorosi, Riccardo Caprari, Teodor Gabriel Crainic, Paolo Dell'Olmo, and Nicoletta Ricciardi. An integrated routing-scheduling model for a hybrid uav-based delivery system. 2020.

[61] Parikshit Maini, Kaarthik Sundar, Mandeep Singh, Sivakumar Rathinam, and PB Sujit. Cooperative aerial–ground vehicle route planning with fuel constraints for coverage applications. *IEEE Transactions on Aerospace and Electronic Systems*, 55(6):3016–3028, 2019.

Appendix A

Related Work Overview

This appendix was created to easily guide the reader in what concerns the literature review contents. Table A.1 presents the different algorithms used in each one of the publications presented in the literature review, while in A.2 the algorithms and publications are categorized regarding their organizational approach.

Algorithms Overview

Algorithms	Publications
Linear Programming	[5], [6]
Particle Swarm Optimization	[7], [8], [9], [10]
Genetic Algorithm	[7], [12]
Greedy Algorithms	[8], [11]
Filter Embedded Task Assignment Algorithm	[14]
MDP	[18], [19]
POMDP	[15], [16]
Consensus Based Bundle Algorithm	[17], [30], [31]

Table A.1: Algorithms Overview

Organizational Paradigms Overview

Approach	Publications
Centralized	[5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [24], [25]
Decentralized	[15], [16], [17], [18], [19], [30], [31]

Table A.2: Organizational Paradigms