Master's Thesis at Aerospace Engineering

A GENETIC ALGORITHM FOR A TASK ALLOCATION PROBLEM IN AN URBAN AIR MOBILITY SCENARIO

Cristiano BICCHIERI

Supervisor: Giorgio GUGLIERI

Co-Supervisor: Stefano PRIMATESTA
Marco RINALDI

April 7, 2023

Abstract

In an aerial package delivery scenario carried out by multiple Unmanned Aerial Vehicles (UAVs), it is important to maximize the collaboration and the resource sharing in the fleet and to satisfy, according to the UAVs' constraints, the largest possible portion of the tasks' requirements' space.
In this thesis work, a UAVs-based parcel delivery problem is formulated and a Genetic Algorithm (GA) is proposed for solving the formulated task assignment problem for a fleet of heterogeneous UAVs.
GAs are a class type of algorithms inspired by the known evolutionary mechanism of populations.
The objective of the proposed task allocation method is to minimize the energy consumed by the fleet for executing the delivery tasks while respecting the time window delivery constraints and the maximum payload capacity of each UAV.
Each task in the task set is considered with the parcel's pick-up point and delivery point, the payload mass, and parcel's time delivery deadline.
The UAVs are assumed to be able to perform only one task at a time. In order to address the service persistency issues, re-charge tasks can also be assigned and added where the UAV's energy is not sufficient to perform the next task.
The initial population of the genetic algorithm is created randomly, and subsequent mutation and crossover operations are performed to increase the diversity of the population and avoid incurring in local minima.
The capability of the proposed solution of efficiently handling the formulated problem is demonstrated with ad-hoc defined scenarios, which represent the algorithm's instances.

Contents

1 Introduction **1**

2 State of the art **3**
2.1 Centralized algorithms 4
2.2 Distributed algorithms 5
2.3 Genetic-based algorithms 6
2.4 Artificial neural network algorithms 7
2.5 Swarm intelligence-based algorithms 8
2.6 Summing up 9

3 Problem statement **10**

4 Proposed method **12**
4.1 Genetic algorithms overview 12
4.2 The procedure of the proposed GA V1 13
4.2.1 Double-chromosome encoding 14
4.2.2 Opposite population 15
4.2.3 Crossover operation 15
4.2.4 Mutation operation 17
4.2.5 Check constraints 19
4.2.6 Add charging tasks 20
4.2.7 Individual evaluation 23
4.2.8 Offspring population creation 24
4.2.9 Best solution selection 25
4.2.10 Add random individuals 25
4.2.11 Stopping criteria 25
4.3 Others versions 26
4.3.1 Mandatory delivery window constraints 26
4.3.2 Charging task only ay the end 27
4.3.3 Final versions 27
4.4 Path planning 28

5 Simulation results **30**
5.1 Simple scenario 31
5.2 Complex scenario 34

6 Conclusions **36**

Bibliography **38**

Nomenclature

δ	Random number generated with $\delta \in [0, 1]$
ϵ	Tolerance below which convergence of the solution can be considered
η	Energy computation efficiency factor
ρ	Air density
ζ	Task on time factor to account for tasks delivered on time
A_d^i	Cross section of UAV i with respect to the direction of motion
A_r^i	Total rotor disk area of UAV i
c_d^i	UAV i drag coefficient
$E_{available}^i$	Current battery energy available in the i-th UAV
$E_{t_j}^i$	Energy used by the UAV i for the j-ish task from the current UAV position
E_{TOT}	Total energy consumed by the N_U UAVs to accomplish the N_T tasks
F_M^i	UA V i Figure of Merit
g	Gravity acceleration
J	Fitting function of the genetic algorithm to be minimized
$L_{1_j}^i$	Minimum-risk path length from UAV i position to delivery task j pick-up position
$L_{2_j}^i$	Minimum-risk path length from payload pick-up to delivery position of task j with UAV i
m^i	UAV i mass
m_{p_j}	Payload mass of delivery task j

mdw Mandatory delivery window parameter

n_i Number of iterations of the GA to reach the solution

N_P Max number of individuals within the population

N_T Tasks number

N_U UAVs number

N_{best} Number of individuals automatiacally chosen for the next generation

n_{char} Number of recharging task in the final solution

n_{conv} Number of the last iterations that are checked for convergence of the solution with the tollerance ϵ

n_{max} Max number of iterations of the GA

$n_{t_{onTime}}$ Number of tasks delivered on time

P Current working population composed by possible solutions as individuals

P' Offspring population for next generation

p_c Crossover probability

p_m Mutation probability

p_{onTime} Percentage of tasks delivered within the time window

P_{OPP} Opposite population created with opposite individuals from P

rc^i_{nearer} Nearer recharging task to i-th UAV position

rce Recharging task only at the end parameter

t^i_j J-th task taken by i-ish UAV

T_{DD_j} Due date of task j

t_{TOT} Total algorithm time

u_i I-th UAV with $i = 0, 1, ..., N_U$

v^i_j I-th UAV velocity to accomplish j-th task

List of Figures

2.1 An example of a simple neural network 8

4.1 Example of the partial-mapped crossover operator. 16
4.2 Example of flip mutation operator. 18
4.3 Example of swap mutation operator. 18
4.4 Example of slide mutation operator. 19

5.1 Operating environment simple scenario 32
5.2 Graph solution of V2 task allocation algorithm for the simple scenario 33
5.3 Progress of the fitting function for each iteration of the V2 algorithm in the simple scenario 34

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs), commonly referred to as drones, have become increasingly popular in recent years due to their versatility and relatively low cost. Drones have a wide range of applications in various industries, including agriculture, surveillance, mapping, and shipping.

In agriculture, drones are used for crop management, such as monitoring crop health and identifying areas that need irrigation or fertilizer. This can help farmers increase their crop yields while reducing the use of resources like water and fertilizer.

In surveillance, drones offer enhanced situational awareness for security and monitoring purposes. They can be used to monitor crowds at events, provide aerial surveillance of large areas, and even aid in search and rescue operations.

In mapping, drones provide high-resolution aerial imaging that can be used to create accurate 3D models of terrain, buildings, and other structures. This information can be used for urban planning, construction, and other applications.

In shipping, drones have the potential to revolutionize the delivery industry. With the rise of e-commerce and the increasing demand for faster and more efficient delivery services, drones have emerged as a potential solution for last-mile delivery. They offer a number of advantages over traditional delivery methods, including faster delivery times, reduced costs, and the ability to reach remote or difficult-to-access areas.

In the delivery sector, drones are being tested and used for a range of applications, including delivering small packages, medical supplies, and even food. Companies such as Amazon, UPS, and Google are investing heavily in drone delivery technology, and several pilot programs have been launched in different parts of the world to test the feasibility of drone delivery.

One of the biggest challenges facing drone delivery is regulatory approval. Govern-

ments around the world are still grappling with how to regulate and manage the use of drones in public airspace, particularly in densely populated areas. There are also technical challenges such as battery life, payload capacity, and weather conditions that must be addressed to make drone delivery a reliable and scalable option.

Despite these challenges, the potential benefits of drone delivery are significant. By reducing the need for delivery trucks and other vehicles, drones have the potential to significantly reduce traffic congestion and carbon emissions, making them a more sustainable and environmentally friendly option. As the technology continues to advance and regulations become more supportive, drone delivery is likely to become a more common sight in the skies in the years to come.

Although problems on maximum flight duration are still evident, a single UAV can complete delivery tasks completely autonomously. As the number of tasks to be accomplished by a UAV increases, it becomes essential to deploy multiple UAVs and thus, it becomes crucial to optimize the distribution of tasks among the UAV fleet.

One key challenge in multi-UAV task assignment is to consider all constraints inherent to the UAVs and tasks while optimizing the distribution of tasks in the fleet. Specifically, the algorithm should aim to minimize energy usage while ensuring delivery times are met. Other constraints may include UAV battery life, payload weight, and the delivery locations and times.

The task assignment problem in a single UAV setting can be viewed as a classic Travelling Salesman Problem (TSP) in the literature, where the objective is to find the shortest path for a salesman to visit all cities. However, increasing constraints significantly complicate the problem, making multi-UAV task assignment an NP-hard combinatorial optimization problem.

To address this challenge, various methods have been proposed in the literature. In chapter 2, we will review these methods and their strengths and weaknesses. In this paper, we propose the use of a genetic algorithm, which utilizes random operations to search for the optimum global solution. This approach has shown promise in solving similar problems and will be evaluated in the subsequent chapters.

Overall, this work aims to optimize the distribution of tasks among a fleet of UAVs, with the goal of minimizing energy usage while meeting delivery times. By doing so, we can enable more efficient and cost-effective deliveries using UAVs.

Chapter 2

State of the art

This chapter provides an overview of the state-of-the-art surveys conducted in the drone delivery system including the challenges and constraints faced.

The drone delivery system can be seen primarily as a task assignment problem, as it contains additional sub-problems such as the routine problem and the charging process.

This is why we will focus on the task assignment issue, and the algorithms used in the literature.

As it is suggested by Poudel and Moh in [16], we can divide the task assignment algorithms in five different categories:

- centralized;
- distributed;
- genetic;
- artificial neural network;
- swarm intelligence.

The last three of them are seen as bio-inspired algorithms, as they are inspired by nature's ability to solve problems.

Generally, the individual drone can be assigned one or multiple tasks, and the goal of the algorithm is to optimize the way tasks are assigned to the different drones that are part of the fleet. This is accomplished by, for example, trying to minimize flight time or consuming the minimum amount of energy possible, all while respecting predetermined delivery times.

Obviously constraints can be added to the problem that add complexity to the calculations. For example, the assigning tasks can be heterogeneous (delivering packages, moving to a charging location, or moving the drone to a place where it is most needed), and similarly we can have a fleet that is heterogeneous, i.e., with different types of drones and therefore with different characteristics, or even different vehicles (e.g., drones in collaboration with trucks).

Similarly, as the number of fleet members increases, so does the complexity of the problem and the time required to solve it.

The ease with which we can make a problem complex means that many of the task-assignment problems can easily become NP-hard optimization problems.

For this very reasons, there is not always a mathematical solution to the problem that leads to the absolute optimum.

Therefore, many solving algorithms set as a goal to find a sub-optimal one, that is one that is not necessarily the absolute optimum of the problem, but still leads to a gain in computational time.

We will now analyze each task assignment algorithms categories with their pros and cons and examples from the literature.

2.1 Centralized algorithms

Centralized task assignment algorithms involve a central hub that performs all the calculations and then appropriately informs the various UAVs.

To allow the central hub to allocate tasks in the best possible way, data exchange from the drones is required for information regarding battery charge or other telemetry data.

Since they rely totally on the central computing computer, any failure of it would lead to a total shutdown of the task allocation system. For this reason, redundancy should be provided to avert any failure.

Problems with few constraints can be viewed as linear type problems and solved as such by arriving at an optimal solution.

Where we begin to add constraints to the problem, however, approximations are needed to arrive at the solution. For example [5] use an approximation to linearize the energy consumption of drones and thus be able to arrive at the optimal solution with even large problems. The fleet is homogeneous and the problem is solved using a branch-and-cut algorithm to solve the linear problem.

Others [8] propose decomposing the problem into two subproblems: the first to account for the cost of shipping on time, and the second to optimize the delivery time from a cost perspective.

They too predict a linear approximation of the consumption problem, based on payload and battery weight, and solve the problem using a mixed linear programming algorithm with suboptimal type solutions.

2.2 Distributed algorithms

Distributed algorithms are based on the equal computation allocation of the task assignement algorithm to each member of the fleet, so each UAV collaborates in the optimal search for the solution to the problem, without having to rely on a central control hub.

These algorithms turn out to be much more robust from the point of view of failure, as by their very nature, each drone is on a par with the others, and therefore any failure of a UAV would not lead to serious consequences. The only consequence will be that the drone will no longer be able to contribute to deliveries until its eventual repair.

Moreover, distributed algorithms are easily scalable in terms of fleet size. Computationally, there is not a large increase in computation time where the number of agents increases.

The most widely used algorithms in the distributed environment are marked-based or auction-based algorithms. These algorithms are based on logic similar to what we have in the marketplace, with each drone that competes with others by bidding to grab the task.

The bid that the drone makes is based on complete knowledge of its telemetry data, including the battery charge level. Data, therefore, that no longer has to be transmitted to the central hub as in the case of centralized algorithms (chapter 2.1).

This implies the need for much less data exchange, but is still vital to have maximum coordination between UAVs.

Obviously, since each drone has its own control center, distributed algorithms require more computational power in the drone's Flight Controller itself when compared with centralized algorithms, but this can be seen as an advantage where one wants to implement dynamic behaviors in response to external events by the drone.

In [10] they use a distributed version of the Hungarian method as a solution algorithm for a task allocation problem. They manage in this way to find a global solution.

As an assumption they have that each component knows the distance between it and the target, but without the use of shared memory or central coordination.

Instead, Oh and Kim, in the paper [13] use a market-based distributed algorithm to make a team of drones cooperate, adding time constraints to the problem. The alogrithm has great scalability capabilities and can also consider dynamic scenarios.

Similarly in [12] use a generalisation of the auction algorithm to solve a similar task allocation problem with time constraints and also considering limited battery life. They take into account a homogeneous bunch of robots but having different suitability for different tasks.

Their algorithm requires little communication requirements and is easily scalable considering that it has a linear trend in solving time as a function of robots and tasks.

2.3 Genetic-based algorithms

Genetic algorithms are an excellent choice for task assigement problems.

They are based on the idea of the heuristic search for natural selection. Starting with a list of candidate solutions, it is found the one (or ones) that best survives the conditions imposed by the problem.

Then starting from this "survivor", mutations and alterations are performed to crate a new population, and this process is iterated until the conditions imposed at the outset are reached.

With genetic algorithms, it is possible to have solutions in extremely fast computation times for even large and complex problems. They can take heterogeneous cases into account with ease but they are not particularly well suited for dynamic scenarios.

In [4] a genetic algorithm is used to optimize the tasks to be apportioned in a fleet of warfare UAVs, with the goal of minimizing their flight time. The tasks must all be executed but some have priority over others.

In contrast, in [23] a genetic algorithm is used to allocate tasks obtaining excellent results especially for large scale problems. An Opposition-based Genetic

Algorithm using Double-chromosomes Encoding and Multiple Mutation Operators (OGA-DEMMO) is used because of the complexity of the problem due to the heterogeneity of the UAV team and tasks set.

Another example of genetic algorithms used to solve complex problems can be seen in [21]. Large fleets of UAVs are considered, taking into account both path and task cost. The genetic algorithm is improved by also using simulated ISAFGA annealing.

Vidal in [3] uses a hybrid genetic algorithm to minimize the total travel time while respecting time constraints and vehicle characteristics.

2.4 Artificial neural network algorithms

Artificial neural network (ANN) algorithms (image 2.1) are gaining momentum in the literature. They are inspired by neurons, specifically by their synapses, the way they process information by creating interconnections between them.

They allow continuous learning that can give extreme adaptability to dynamic environments. Moreover, with these algorithms there is generally better optimization at the global level and not just in the locale of the specific moment.

ANN algorithms can be extremely fast and computationally efficient, taking into account complex and heterogeneous problems.

The downside of these algorithms is that a large dataset with multiple different scenarios is required for learning to occur appropriately.

In fact, ANN algorithms do not work well in cases where scoring population is significantly different compared to training sample.

For example, in [22] could be seen how a neural network algorithm is used to reallocate tasks to a team of robots dynamically.

Another interesting example is [19] where multi-agent reinforcement learning (RL) is used to solve a dynamic task allocation problem for vehicles in an urban environment.

RL is a special method of ANN. Learning does not occur from a database but the algorithm itself learns automatically in a try-and-fail way that involves awarding a prize when the algorithm has donated a good solution.

Although it requires a longer initial computational time in the learning phase, the algorithm donates excellent results even in dynamic environments.

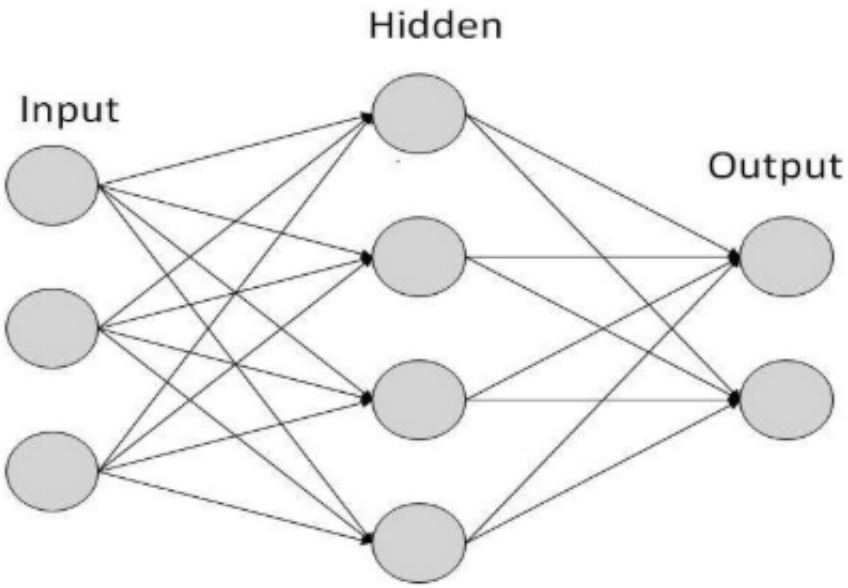

Figure 2.1: An example of a simple neural network

2.5 Swarm intelligence-based algorithms

Swarm intelligence algorithms imitate the behavior of teams of simple enities working together toward a common goal.

Through interaction with the surroundings and other members of the group, a collective consciousness is generated that allows the problem to be solved effectively which is completely impossible for each member of the group.

Such behavior we can find for example in ant colonies or bee colonies, but it is present both in larger animals, where they group together in herds, and in microorganisms such as bacteria.

Algorithms of this kind can be easily scalable and well robust, with short computation times. In particular, they can prove to be very flexible to easily adapt to new scenarios and react to changes in the environment.

In [9] a K-means algorithm is used to assign tasks to a team of drones and then use a hybrid algorithm based on bionic learning to assign tasks. The goal is to minimize time and energy consumption.

The authors say that with not too many drones the algorithm gives better results than other algorithms they tested, including a gentic algorithm.

2.6 Summing up

To conclude this chapter on the state of the art, we can say that does not exist a better solution for task assignements problems.

The algorithm to be used depends very much on the needs and means at disposal.

Obviously if you have a large amount of data and great computational power, neural networks algorithms are recommended as they give better results in less time and computational cost.

It should be noted, however, that the fusion of two or more different algorithms is possible, and indeed recommended, to overcome the problems inherent in only one type of algorithm and thus obtain more robust results that take into account more variables at the same time.

Furthermore, it is important to note that this chapter is not a complete and exhaustive survey of the state of the art as this is not the purpose of this work. For any further study, the reader is invited to more appropriate texts such as [16] and [2].

Chapter 3

Problem statement

This chapter aims to give a clearer idea of the problem adressed and donate the context the algorithm operates.

The main problem is to allocate N_T delivery tasks on a team of N_U UAVs. Each delivery task involves the delivery of a parcel with mass m_p, from a pickup location P_1 to a delivery location P_2.

The goal is to allocate all charging tasks, minimizing the total energy use E_{TOT} required by the fleet of UAVs to complete all tasks, while trying to meet the time delivery window T_{DW} of all tasks.

Nowadays, the operational capacity of a battery is the most restrictive constraint in drone utilization, so charging tasks are allocated, where they are needed by the drone to complete the remaining tasks assigned to it.

In this way we ensure the continuous operation of the drone, which could be reused for a new allocation at the end of the mission given to it.

Recharging tasks are defined by a P_{RT} location within the operational area, upon reaching which the drone's battery is changed in negligible time.

As simplifying assumptions to the problem we then have:

- The battery change time is considered to be zero.
- During the execution of the task, the speed of the UAV remains constant.
- The flight altitude is considered constant.
- A UAV can take charge of only one task at a time, after which it will move on to the next one.
- The trajectory of each UAV is simplified as straight.

- Any weather events such as wind and gusts are not taken into account.
- Takeoff and landing moments are not considered in the energy calculation.

As we will see later in chapter 4, the ability to operate in 4 different scenarios has been added to the algorithm, based on two parameters that are chosen a priori.

The energy calculation for adding the charging tasks leads to a significant slowdown of the algorithm, so through a *rc only at the end* parameter it is possible to choose whether to apply the calculation of them only at the end, once the solution is found. This will certainly lead to further simplification of the problem, but to the benefit of reduced computation time.

Regarding task delivery times, they can be considered as hard or soft constraints, through the *mandatory delivery window* parameter.

In this way, for complex scenarios, where it would not be possible to complete all tasks in the due time, the algorithm can still arrive at a solution, if the constraint is set as soft. The solution found in this way will always be the one with the lowest energy E_{TOT} but which also respects the largest number of delivery windows.

The two parameters can be combined, thus resulting in 4 different possible solutions, starting from the same operational scenario.

We will look at these features and operations in more detail in chapter 4, with examples on the different solutions found in the following chapter 5,.

Chapter 4

Proposed method

In this chapter we will look in detail at the algorithm used to solve the task assignment problem.

It is part of the family of genetic algorithms and as such, aims to arrive at a solution using a set of random operations.

4.1 Genetic algorithms overview

As seen in the previous chapter, a genetic algorithm (GA) is a type of optimization algorithm inspired by the process of natural selection and genetics. It starts with a population of potential solutions to a problem, and iteratively improves them by applying a set of genetic operators such as selection, crossover, and mutation.

The process begins with the creation of an initial population of potential solutions. Each solution is represented as a string of values, called chromosomes, which can be thought of as the candidate solution's genes. These chromosomes are evaluated according to some fitness function, which determines how well they solve the problem.

The genetic operators then begin to act on the population. The selection operator chooses the fittest individuals from the population and allows them to reproduce by generating offspring. The crossover operator combines the chromosomes of two individuals to create a new individual, while the mutation operator modifies an individual's chromosome in a random way to introduce new genetic material.

The new population of individuals produced by these operators is then evaluated, and the process repeats until a satisfactory solution is found or some stopping criterion is met.

The reason why GAs are well-suited to solve NP-hard problems is that they are able to search a large and complex solution space efficiently. NP-hard problems are those that cannot be solved in polynomial time, and traditional optimization techniques such as brute force or gradient-based methods become computationally infeasible as the problem size increases.

GAs are able to effectively search large and complex solution spaces by exploring multiple solutions simultaneously and avoiding getting stuck in local optima. The ability to maintain a diverse population of potential solutions also helps prevent premature convergence to suboptimal solutions.

Overall, GAs are a powerful and flexible optimization tool that can be applied to a wide range of problems, including those that are NP-hard.

4.2 The procedure of the proposed GA V1

The genetic algorithm proposed for task allocation can be found in Algorithm 1.

For semplicity reason here we will see only the version 1 (V1) of the algorithm. This version is obtained with the configuration previously discussed in Chapter 3, where the *mandatory delivery window* variable is set to `true` and *rc only at the end* is set to `false`. However, it is important to note that different versions of the algorithm exist, with varying combinations of variable values. These will be explored in the next section (Section 4.3).

The genetic algorithm presented is designed for task allocation with a double chromosome encoding (Section 4.2.1). The algorithm starts by creating a random population, denoted as P, consisting of N_P individuals. The algorithm then iterates through a loop until a stopping criteria is met (Section 4.2.11).

In a genetic algorithm, an individual is a potential solution to the problem being solved. It is represented by two list of integers, defined as chromosomes, which encodes the candidate solution's properties.

During each iteration, an opposite population, denoted as P_{OPP}, is created from P using an opposition-based learning approach (Section 4.2.2). Crossover and mutation operations (respectively Section 4.2.3 and Section 4.2.4) are then performed on both P and P_{OPP} to generate new individuals. The max payload constraint is checked in both populations (Section 4.2.5), and charging tasks are added where needed (Section 4.2.6). The delivery window constraint is then checked in both populations (Section 4.2.5). At this point, each individual in both populations (P

Algorithm 1 Task allocation GA proposed - V1

```
1: procedure START GA
2:     Random population P creation with N_P individuals
3:     while Stopping criteria is not meet do
4:         Opposite population P_OPP creation from P
5:         Crossover operation in P and P_OPP
6:         Mutation operation in P and P_OPP
7:         Check max payload constraint in P and P_OPP
8:         Add charging tasks in P and P_OPP
9:         Check delivery window constraint in P and P_OPP
10:        Populations (P and P_OPP) evaluation
11:        Offspring population P' creation from P and P_OPP
12:        Best solution selection
13:        while Individuals in P' < N_P do
14:            Add random individual in P'
15:        end while
16:        P = P'
17:    end while
18: end procedure
```

and P_{OPP}) is evaluated (Section 4.2.7).

A new population offspring P' is created by combining the individuals from P and P_{OPP} (Section 4.2.8). The best solution is then selected from P', and a while loop is entered to ensure that the population size remains constant. During this loop, new individuals are added to P' until it contains exactly N_P individuals. Finally, P is replaced with P' for the next generation operations.

This process continues until the stopping criteria is met at which point the best solution is returned. Overall, the genetic algorithm employs several random operations during each iteration to search for a solution. This approach may reduce the risk of getting trapped in local optima but does not guarantee that the global optimal solution will be found.

4.2.1 Double-chromosome encoding

In the context of genetic algorithms, the chromosome is a representation of the solution to the optimization problem at hand. The chromosome is typically encoded as an array of integers, which can be modified through genetic operators like crossover and mutation.

For the present study, a double-chromosome encoding approach was employed. This type of encoding is commonly utilized in genetic algorithms for optimization problems that involve multiple variables.

In the double-chromosome encoding scheme adopted in this work, the first chromosome (chromosome I) represents the task delivery sequence, while the second chromosome (chromosome II) encodes the cut positions of the task delivery sequence in chromosome I.

In chromosome I, each gene represents the index of a delivery task To ensure the validity of the solution, the genes in chromosome I must be unique, and the total number of genes is equal to N_T. Chromosome II, on the other hand, encodes the cut positions of chromosome I. Specifically, the value of any gene in chromosome II must not be smaller than the values of the genes that precede it. Additionally, the number of genes in chromosome II is set to $N_U - 1$ to ensure that the task delivery sequence in chromosome I is partitioned into N_U subsequences.

4.2.2 Opposite population

Opposition-based learning in genetic algorithm is founded on the observation that the opposite of a weak attribute is strong in the real world. This concept is utilized to generate an opposite population P_{OPP}, following the initialization operation, to explore broader region of the solution space.

For a variable z in the range $[a, b]$, its opposition z' is defined as:

$$z' = a + b - z \tag{4.1}$$

In the GA proposed, each integer of chromosome I of an individual is treated as a variable to calculate its opposition. And so in our case, considering z_i the i-th element of the array of a choromosome I, its opposite will be:

$$z'_i = a + b - z_i \qquad i \in [0, N_T] \tag{4.2}$$

In this particular task allocation problem, $a = 0$ and $b = N_T$. For instance, if chromosome I is (1, 7, 0, 4, 5, 3, 6, 2), with lower and upper bounds of 0 and 7, respectively, then the opposition of chromosome I would be (6, 0, 7, 3, 2, 4, 1, 5).

4.2.3 Crossover operation

This algorithm employs a crossover operator to create a pair of offspring chromosomes from a pair of parent chromosomes. However, it is worth noting that only

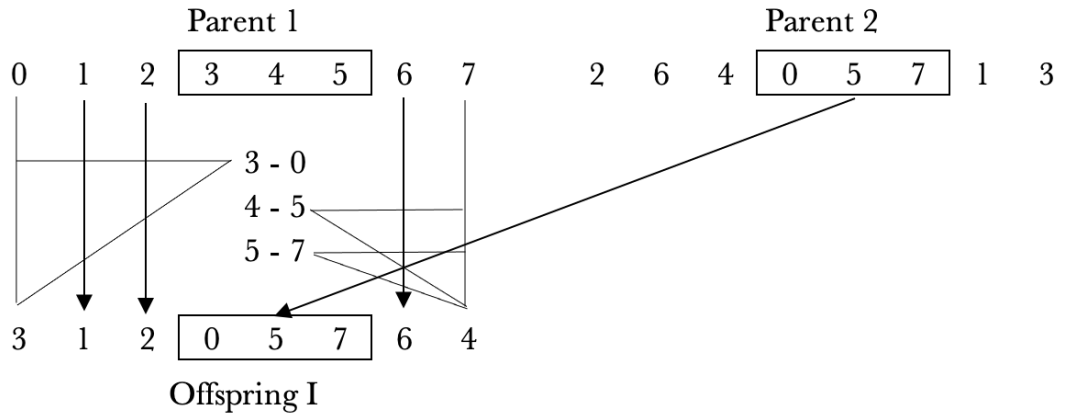

Figure 4.1: Example of the partial-mapped crossover operator.

chromosome I undergoes the crossover operation, while chromosome II remains unchanged.

The partial-mapped crossover (PMX) operator is utilized in this work, where a portion of genes from one parent is exchanged with the corresponding genes of the other parent while preserving the remaining genes.

Algorithm 2 Crossover operation in population P

```
1: procedure START CROSSOVER IN P
2:     for each individual P_i in P, with i = 0, 1, ..., N_P do
3:         if δ < p_c then
4:             Perform crossover in P_i with P_j (j ≠ i randomly chosen)
5:         end if
6:     end for
7: end procedure
```

Chromosome II is chosen at random in the population, making sure that chromosome I is different from chromosome II.

Algorithm 2 demonstrates that the crossover operation is not applied to all individuals in the population, but only to those for which the randomly generated number δ is less than the crossover probability parameter p_c.

For each individual in the population, a random number $\delta \in [0, 1]$ is generated, and if it is less than the value of p_c, then the individual undergoes the crossover operation.

It is worth noting how the same principle is then applied to all individuals in the population P_{OPP}.

To illustrate the process of PMX, Figure 4.1 shows that two cut points are randomly selected between the third and fourth genes and between the sixth and seventh genes. Two mapping sections, (3, 4, 5) and (0, 5, 7), are identified along with the mappings 3-0, 4-5, and 5-7. The mapping sections from parent I and parent II are then copied to offspring II and offspring I, respectively. The remaining genes in the offspring chromosomes are filled up by copying from the corresponding parents or regenerating through the mappings.

For example, the first gene of offspring I is initially copied from parent I, but since the gene 0 already exists, it is reset to 3 according to the mapping 3-0. Similarly, the last gene of offspring I is 7 when copied from parent I, but since it already exists, it is mapped to 4 according to the mappings 4-5 and 5-8, resulting in the offspring I chromosome (3, 1, 2, 0, 5, 7, 6, 4).

4.2.4 Mutation operation

Mutation operations play a crucial role in preventing a population from getting stuck in local minima and improving the overall convergence of a GA algorithm. The GA proposed utilizes multiple types of mutation operators to increase population diversity and enhance global exploration.

As seen on Algorithm3, the GA proposed, after the crossover operation, the population is divided into groups of eight individuals, and the best individual of each group is mutated to generate eight new individuals, which will be then added to the respective population.

This is done, following the same principle seen in crossover (Section 4.2.3). Only where the random value δ results in less than the mutation probability p_m will the group of 8 individuals undergo mutation.

The same logic will then be used in the population P_{OPP}.

Because of the double-chromosome encoding used, different mutation operators are applied to each chromosome.

Chromosome I is subject to flip, swap, and slide mutation operators, while chromosome II undergoes a regenerate mutation operator. By combining these operators, eight mutation operation results can be produced for double-chromosome encoding.

. For example, if chromosome I is (1, 2, 3, 4, 5, 6) and chromosome II is (2, 4), eight offspring individuals can be generated using different mutation operations.

Algorithm 3 Mutation operation in population P

```
1: procedure START MUTATION IN P
2:     for each group of 8 individuals [P_i, P_{i+8}] in P do
3:         if δ < p_m then
4:             Perform mutation in [P_i, P_{i+8}]
5:         end if
6:     end for
7: end procedure
```

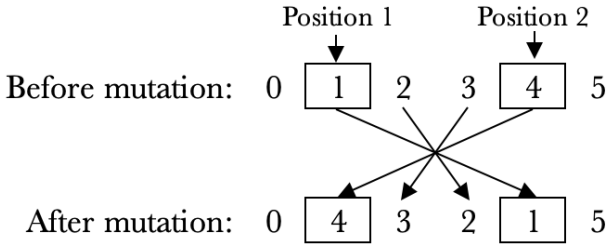

Figure 4.2: Example of flip mutation operator.

Flip mutation

To perform the flip mutation operator, the algorithm randomly selects two positions on chromosome I. The genes between these two positions are then reversed. An illustration of this process is provided in Figure 4.2.

Swap mutation

The mutation process for chromosome I involves randomly selecting two positions, followed by exchanging the genes at these positions, as illustrated in Figure 4.3.

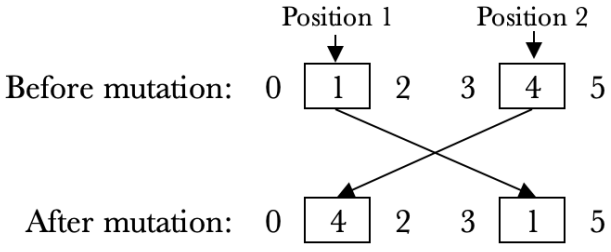

Figure 4.3: Example of swap mutation operator.

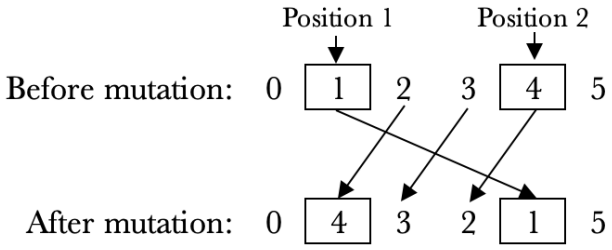

Figure 4.4: Example of slide mutation operator.

Slide mutation

Firstly, two positions are randomly selected on chromosome I. Next, the gene on the first selected position is moved to the end of the selected substring, causing the remaining genes of the substring to slide forward. An example of the slide mutation operator can be seen in Figure 4.4.

Regenerate mutation

The regenerate mutation operator randomly regenerates each gene of chromosome II while ensuring that the constraints of chromosome II are satisfied during the process.

4.2.5 Check constraints

To ensure that the final solution satisfies the constraints imposed by the problem, each individual in the populations P and P_{OPP} undergoes a feasibility check.

Before undergoing selection for the next generation population, the individuals are verified for their feasibility in terms of the maximum payload that can be carried by each UAV and the delivery windows considering the added recharging tasks. While the payload constraint is considered hard, meaning that all solutions must respect it, the delivery time constraint can be either hard or soft depending on the parameter *mandatory delivery window* (*mdw)*. The details of this constraint as a soft constraint are discussed in Section 4.3.

The feasibility check is performed after mutation and crossover operations, which increase the diversity and randomness of individuals in the populations. This approach allows for the possibility of finding usable individuals even when starting from nonusable ones.

Individuals that do not satisfy the constraints are removed from the respective

population and do not undergo the subsequent selection process as described in Section 4.2.8.

Check max payload

After the crossover and mutation tasks, each individual in the population undergoes a feasibility check to ensure that it satisfies the maximum payload constraint. Specifically, the feasibility check verifies that each UAV in the solution is assigned tasks that do not exceed the UAV's maximum payload capacity.

Infeasible individuals are discarded, thereby reducing the computational burden of generating recharge tasks. As described in Section 4.2.6, generating recharge tasks can be computationally expensive. Thus, removing infeasible individuals can significantly reduce the time required to add recharge tasks.

Check delivery window

The constraint on delivery time is checked only after the addition of reloading tasks, to account for the time taken to reach those locations.

Each individual in the population is then checked to ensure that the time taken by the UAV to execute each task is within the specified delivery window for that task, taking into account the time taken by the UAV to reach the initial position.

This is accomplished by calculating the energy used by the drone and the corresponding speed and travel time needed to execute the tasks, as explained in detail in Section 4.2.6.

4.2.6 Add charging tasks

After crossover and mutation operations, individuals that satisfy the maximum payload constraint undergo the addition of recharging tasks, in order to enable them to recharge their batteries and complete remaining deliveries. The algorithm assumes that all tasks and their order are known in advance.

The algorithm used to add charging tasks can be seen in Algorithm 4. A more specific overview on the optimizer and the path planner can be seen in the next sections

When recharging tasks are only added at the end, the same algorithm is used with a slight modification. In this case, the algorithm will only be called at the end, after all iterations have been performed, and the individual in input to the algorithm will be the solution itself.

Algorithm 4 Add charging task to individual I

```
1: procedure START
2:     Decode C I and C II of I in task list per UAV
3:     for each uav u_i with i = 0, 1, ..., N_U do
4:         for each task t_j^i in task list of UAV u_i do
5:             Compute E_{t_j}^i by calling optimizer and path planner
6:             if E_{t_j}^i > E_{available}^i then
7:                 Find nearer recharging task rc_{nearer}^i to current u_i position
8:                 Add rc_{nearer}^i before t_j^i
9:             end if
10:        end for
11:    end for
12: end procedure
```

To add charging tasks, the individual's two chromosomes are decoded to create a task list for each UAV. Only regular tasks are present in chromosome I, and recharging tasks are added solely to ensure a feasible solution. The recharging tasks are not considered in the subsequent rounds of the genetic algorithm and are not subject to crossover and mutation operations.

For each UAV, the algorithm runs through each task in the task list one at a time, and after calculating the energy required for the task to be completed, it checks whether there is sufficient energy in the UAV's battery. If the battery level is sufficient, the energy required for the task is subtracted from the battery level, and the position of the UAV is updated with the final position of the task. If the battery level is insufficient, a recharge task is added before taking the task, and the position is updated accordingly.

If at any time during the algorithm, a UAV has a fully charged battery, and the energy required to perform the next task (charging or not) is higher than the battery capacity, the individual I is discarded because it does not lead to a feasible solution.

Algorithm 4 starts by decoding chromosomes I and II of individual I into task lists per UAV. For each UAV, the algorithm then loops through the task list and checks whether the energy required for each task is greater than the energy available in the UAV's battery. If so, the algorithm finds the nearest recharging task to the UAV's current position and adds it to the task list before the task that requires recharging.

In summary, the algorithm for adding charging tasks ensures that UAVs have sufficient energy to complete all tasks by adding recharging tasks to the task list. and trasform the double chromosome into a feasible solution This approach allows for efficient and effective task allocation in UAV delivery systems.

The energy calculation for each task is performed through the use of optimizers that differ depending on whether the task is delivery or charging.

However in both cases the optimization find the energy demanded by the task and the time needed to complete it. These values will then be used in the fitting function (Eq. 4.8) and during the phase of check constraints.

Task energy calculation

In the case of delivery tasks, the optimizer aims to minimize the energy used by the UAV:

$$min \quad E^i_{t_j}(v^i_j) = \frac{1}{\eta}\Bigg(c^i_d \rho A^i_d (L^i_{1_j} + L^i_{2_j}) v^{i^2}_j + \\ + \frac{L^i_{1_j}\sqrt{\left((m^i + m_{p_j})g\right)^3}}{v^i_j F^i_M \sqrt{2\rho A^i_r}} + \frac{L^i_{2_j}\sqrt{(m^i g)^3}}{v^i_j F^i_M \sqrt{2\rho A^i_r}} \Bigg) \tag{4.3}$$

Subject to the following constraints:

$$0 \leq E^i_{t_j}(v^i_j) \leq E^i_{available} \tag{4.4}$$

$$0 \leq v^i_j \leq v^i_{MAX} \tag{4.5}$$

$$0 \leq T^i_i + \frac{L^i_{1_j} + L^i_{2_j}}{v^i_j} \leq T_{DD_j} \tag{4.6}$$

Equation 4.3 represents the energy consumption required by UAV i to perform task j with velocity v^i_j. This energy consumption model is adapted from the work of Aiello et al. [1], which proposes an energy estimation method for UAS-based urban logistics based on Newton's equilibrium in steady-state flight conditions.

However, it should be noted that this model does not take into account factors such as cross-section variations during flight, energy consumption during take-off and landing, changes in airflow direction, or different UAV speeds during task execution. Nevertheless, given our assumptions, this model provides a satisfactory estimation.

The constraint on the minimum remaining energy after task completion is expressed in Equation 4.4. The velocity of the UAV during the task is subject to constraints on its upper and lower bounds, as stated in Equation 4.5. Furthermore, the task's due date constraint is represented by Equation 4.6.

The parameters L_{1_j} and L_{2_j}, as we will see in Section 4.4, are derived by calling the path planner, to have the minimum risk path.

Recharging task energy calculation

When it comes to a charging task, it is no longer necessary to have the speed that minimizes energy, but rather that the travel time is as minimal as possible.

In fact we can finish all the energy left inside the battery so that the time lost per recharge is the minimum.

In this case then the equation to be minimized will be as follows:

$$min \quad t^i_{t_j}(v^i_j) = \frac{L^i_{1_j} + L^i_{2_j}}{v^i_j} \tag{4.7}$$

Subject to the Eq. 4.4 and Eq 4.5 constraints, with the parameters L_{2_j} and m_{p_j} equal to 0.

4.2.7 Individual evaluation

Individual evaluation is a critical step in the genetic algorithm optimization process. During this step, each individual in the population is assessed based on its fitness or objective function value.

The fitness function represents the quality of the solution encoded in the individual's chromosome.

The fitness function is problem-specific and must be designed to evaluate the individual's solution to the problem at hand. It should take into account all the constraints and goals of the problem and return a score that reflects how well the individual's solution meets these criteria.

Once each individual's fitness value is calculated, they can be ranked according to their scores, and the best solutions can be selected to form the next generation.

This selection process can be based on different strategies, such as elitism, where the best individuals are directly copied to the next generation, or tournament selection, where individuals are randomly selected and compared in pairs.

In summary, individual evaluation is a crucial step in the genetic algorithm optimization process. It involves assessing each individual's fitness based on a problem-specific fitness function and ranking them to select the best solutions for the next generation.

Specifically, in the proposed task allocation algorithm, the fitting function is as follows:

$$min \quad J = \zeta \sum_{i}^{N_U} E^i_{TOT} \tag{4.8}$$

that is, the sum of the total energy used by each drone to perform all the assigned tasks. In fact, for the individual drone we can write:

$$E^i_{TOT} = \sum_{j}^{N_T} E^i_{t_j} \tag{4.9}$$

The parameter ζ in the Equation 4.8 in the specific case of the Algorithm 1 is constant and equal to 1. This is because it takes into account how many tasks do not comply with the delivery window and in the specific case of *mandatory delivery window* equal to `true` is not taken into account.

More informations about this parameter could be seen in the Section **??**.

4.2.8 Offspring population creation

To generate the next generation, an offspring population is created by merging individuals from both P and P_{OPP}. Specifically, half of the individuals are randomly selected from P, and the remaining half from P_{OPP}. The N_{best} individuals with the highest fitness scores from both populations are automatically included in the new population P'.

For the remaining individuals, the roulette wheel selection technique is applied. This selection method is widely used in genetic algorithms and involves assigning each individual a selection probability proportional to its fitness score.

The advantage of roulette wheel selection is that it favors individuals with higher fitness values, while still allowing individuals with lower fitness values to have a chance of being selected. This helps to maintain diversity in the population and prevent premature convergence.

Additionally, it is a simple and easy-to-implement method that does not require any additional parameters beyond the fitness values and it ensures that the new

population has a higher overall fitness than the previous generation. However, it can be slow for large populations or if there is a wide range of fitness values, as it requires calculating the fitness values and probabilities for all individuals in the population.

4.2.9 Best solution selection

After generating the offspring population, the algorithm proceeds to select the individual with the lowest fitness function from the list of individuals. This selected individual forms the solution for the current iteration and is stored separately.

This step is crucial because Section 4.2.11 demonstrates that the algorithm leverages the history of past solutions to determine when to stop if it converges.

Additionally, this approach safeguards against losing the best solution as the chance-based crossover and mutation operations could lead to the displacement of the individual with the highest fitness function, thereby removing it from the P population.

4.2.10 Add random individuals

To maintain a constant starting population size (P) during the genetic algorithm, a check is conducted to ensure that the population doesn't fall below the pre-specified population size (N_P) before transferring individuals from the offspring population (P') to P.

If P' has fewer individuals than N_P, new individuals are randomly generated and added to P' until N_P is reached. This approach helps maintain diversity in the population, as a lack of diversity can lead to premature convergence and suboptimal solutions.

By introducing new individuals via random generation, previously unexplored search spaces can be explored, and the probability of discovering new candidate solutions is increased.

4.2.11 Stopping criteria

The algorithm stops when one of the following two cases is satisfied:

- The number of iterations is equal to the maximum iteration number $n = n_{max}$.
- We have convergence of the solution and thus in the last n_{conv} iterations, the solution remains below a tolerance ϵ.

4.3 Others versions

The algorithm we have discussed thus far pertains to version 1, which can be customized using the parameters *mandatory delivery window* and *rc only at the end*. By adjusting these parameters to `true` or `false`, we can create up to four distinct versions of the allocation algorithm.

It should be noted that the different versions may not necessarily produce different final solutions, as the optimal solution for one version may be the same as that for another, given that the fitness function remains unchanged.

4.3.1 Mandatory delivery window constraints

We can choose whether to treat delivery windows as hard or soft constraints using the parameter *mandatory delivery window*, where setting it to `true` or `false` corresponds to hard or soft constraints, respectively. For hard constraints, any solution that fails to adhere to the delivery window for a given task is disregarded. Algorithm 1 follows this approach.

Soft constraints, on the other hand, do not discard solutions that do not satisfy the delivery window, but instead disfavor them.

Let p_{onTime} represent the percentage of tasks delivered within the window, then:

$$p_{onTime} = \frac{n_{t_{onTime}}}{N_T} \tag{4.10}$$

Where with $n_{t_{onTime}}$ we indicate the number of tasks delivered on time.
The parameter ζ in Equation 4.8 is modified as follows:

$$\zeta = \frac{1}{p_{onTime}} \tag{4.11}$$

The fitting function in the proposed algorithm is adjusted based on the percentage of tasks delivered within their respective scheduled delivery windows. When all tasks are completed within their scheduled windows, the fitting function is multiplied by a value of one. However, as the number of tasks not completed within their scheduled windows increases, the value by which the fitting function is multiplied also increases.

As a result, the algorithm may prioritize a task allocation that minimizes energy consumption but fails to adhere to delivery windows, over an allocation that consumes more energy but satisfies all delivery windows.

4.3.2 Charging task only ay the end

The computational time required for incorporating charging tasks is substantial, as it necessitates the optimizer to calculate energy and travel times. To address this, we introduced the parameter *rc only at the end*. When set to `true`, the recharge tasks are added after the final solution is found, allowing for faster completion of the algorithm. However, this approach sacrifices accuracy as the algorithm may not discover the global minimum without considering recharge tasks.

The modified Algorithm 1 searches for the energy required for each drone for each task, without considering the drone battery's charge state, which is assumed to be maximum. The resulting version of the algorithm can be seen on Algorithm 5.

Algorithm 5 Task allocation GA with recharging task only at the end

```
1: procedure START GA
2:     Random population P creation with N_P individuals
3:     while Stopping criteria is not meet do
4:         Opposite population P_OPP creation from P
5:         Crossover operation in P and P_OPP
6:         Mutation operation in P and P_OPP
7:         Check max payload constraint in P and P_OPP
8:         Energy calculation in P and P_OPP
9:         Compute ζ for each individual in P and P_OPP
10:        Populations (P and P_OPP) evaluation
11:        Offspring population P' creation from P and P_OPP
12:        Best solution selection
13:        while Individuals in P' < N_P do
14:            Add random individual in P'
15:        end while
16:        P = P'
17:    end while
18:    Add recharging task to final solution
19: end procedure
```

4.3.3 Final versions

The final versions of the algorithm can be classified into four distinct types based on the parameter settings for *mandatory delivery window* (*mdw*) and *rc only at the end* (*rce*). In this context, the parameter mdw determines whether the algorithm should consider the delivery window as a hard or soft constraint, whereas

rce decides whether the recharging tasks should be added during the optimization process or only at the end.

The resulting versions of the algorithm are denoted as V1, V2, V3, and V4.

Specifically, when both *mdw* and *rce* are set to `true`, the algorithm takes the form of V3. When *mdw* is `true` and *rce* is `false`, the algorithm is referred to as V1. When *mdw* is `false` and *rce* is `true`, the algorithm takes the form of V4. Finally, when both *mdw* and *rce* are `false`, the algorithm is denoted as V2.

A schematic view of the different versions can be had from the Table 4.1.

mdw / rce	T	F
T	V3	V1
F	V4	V2

Table 4.1: Task allocation alogithm version schema

4.4 Path planning

In the proposed evolutionary-based task allocation solutions, multiple paths connecting the UAV position to the task positions need to be computed. To achieve this, the risk-aware path planning method proposed in Primatesta et al. [18] is adopted. This method involves a two-step procedure where a risk map is first generated, followed by a path planning algorithm that searches for the minimum risk path while minimizing the overall risk and flight time.

As described in Primatesta et al. [17], the risk map is a two-dimensional location-based map, where each element represents a specific location and is associated with a risk value. The risk value is computed using a probabilistic ground risk assessment approach that estimates the expected frequency of fatalities after a ground impact accident, expressed in fatalities per flight hour (h^{-1}). The risk map depends on the UAV type and characteristics, such as mass, dimensions, and maximum flight speed. Thus, a risk map must be computed for each UAV type considered, including the mass of the payload delivered. For more details about the generation of the ground risk map, please refer to Primatesta et al. [17].

After generating the risk map, the risk-aware path planning approach utilizes the RRT* algorithm, introduced by Karaman and Frazzoli [11], to compute the minimum risk path in the map. RRT* explores the search space, constructing an asymptotically optimal tree, and the near-optimal solution is the branch of the tree that connects the start and goal. The method is used to minimize overall

risk with respect to flight time, where the risk, expressed in flight hour (h^{-1}), is proportional to the flight time. For more information about the risk-aware path planning approach, please refer to Primatesta et al. [18].

Moreover, before returning the path to the task allocator, it is important to verify that the average risk of the minimum risk path is lower than an Equivalent Level of Safety (ELOS), as recommended by Dalamagkidis et al. [7]. An acceptable ELOS with lightweight UAVs is $10^{-6}h^{-1}$. This last step is crucial because the risk-aware path planning approach only returns the minimum risk path in the risk map, without ensuring that the computed path has an adequate level of safety.

The path planner generates output in the form of path lengths, which are then utilized as parameters of the tasks $L1$ and $L2$ in Equation 4.3 and given to the optimizers for further processing.

Chapter 5

Simulation results

This chapter presents numerical simulations that showcase the effectiveness of the proposed method and established models.

The algorithms are implemented in Python 3 and tested in three different scenarios. The first scenario is a simple test to verify the algorithm's proper functioning and to provide a clear understanding of its operation. The remaining two scenarios are complex, representing real-world situations with multiple tasks and UAVs for task allocation.

To compare the results obtained, each scenario was tested using all four versions of the algorithm. For the simulations, a constant UAV thrust-to-weight ratio F_M of 0.9, c_d of 0.3, η of 0.8, and a density of ρ of 1.23 kg/m^3 were assumed.

Table 5.1 provides a list of further characteristics of the fleet of UAVs used, with a maximum payload equal to the mass of the UAV itself $m^i_{p_{MAX}} = m^i$ for all of them.

The tasks were created from a set of 40 points in the operational environment, with each task having a pickup point and an end point.

The payloads were assigned to have 4 different weights: 0.5, 1, 2, 3, and 4 kg. Additionally, four charging points were defined in the operational environment.

The operational environment used was a portion of the city of Turin, Italy, and the path planner was launched on it accordingly.

To reduce computational time, the path planner was launched prior to the task allocation algorithm. The path planner calculated the values of the distances between the various points and saved them in a matrix (triangular with zero diagonal) that the algorithm retrieved when it required data regarding the distance between two points, instead of recalculating it.

UAV id	$m[kg]$	$A_r[m^2]$	$A_d[m^2]$	$v_{MAX}[m/s]$	$E_{MAX}[MJ]$
0	1	0.2	0.4	16	0.68
1	2	0.28	0.6	19	0.9
2	3	0.36	0.8	20	1.17
3	4	0.44	1	22	1.43

Table 5.1: UAVs' fleet characteristic

In both the simple and complex scenarios, the genetic algorithm was configured with values of 0.3 for both the mutation and crossover probabilities (p_m and p_c), a population size of 30 individuals (N_P), the number of elite individuals selected for the next generation set to 5 (N_{best}), the convergence threshold (n_{conv}) set to 8, the energy tolerance between the last solutions (ϵ) set to 300 J, and the maximum number of iterations of the GA (n_{max}) set to 20.

5.1 Simple scenario

For the simple scenario, the task is to allocate 5 delivery tasks ($N_T = 5$) among a fleet of 2 UAVs ($N_U = 2$), with the availability of 2 charging stations ($N_{CT} = 2$) for the UAVs to use when required. The UAVs used in this scenario are identified as number 1 and number 3, as shown in Table 5.1.

The tasks were randomly selected from the list of 40 points, as were the charging stations. Each task's delivery window (T_{DD_j}) was assigned randomly from 0 to 3 hours.

The operating environment is illustrated in Figure 5.1, which presents the complete scenario. The 5 delivery tasks are indicated with a continuous line, and the pickup position is denoted by a bottom-up triangle while the delivery position is marked with a top-down triangle, as shown in the legend. The 2 recharging stations are represented by a "+" symbol, and the starting points of the two UAVs are shown on the map with a square.

The scenario is solved using the 4 different versions of the algorithm and the results are shown in Table 5.2.

We can visualize the solution obtained by version 2 of the algorithm on the map in Figure 5.2. The path taken by the drone starting from the initial position is shown with a dotted line. UAV 0 performs tasks 1, 0, and 4 before heading to the charging station (6) and then completing the last task 3. On the other hand, UAV 1 only handles task 2.

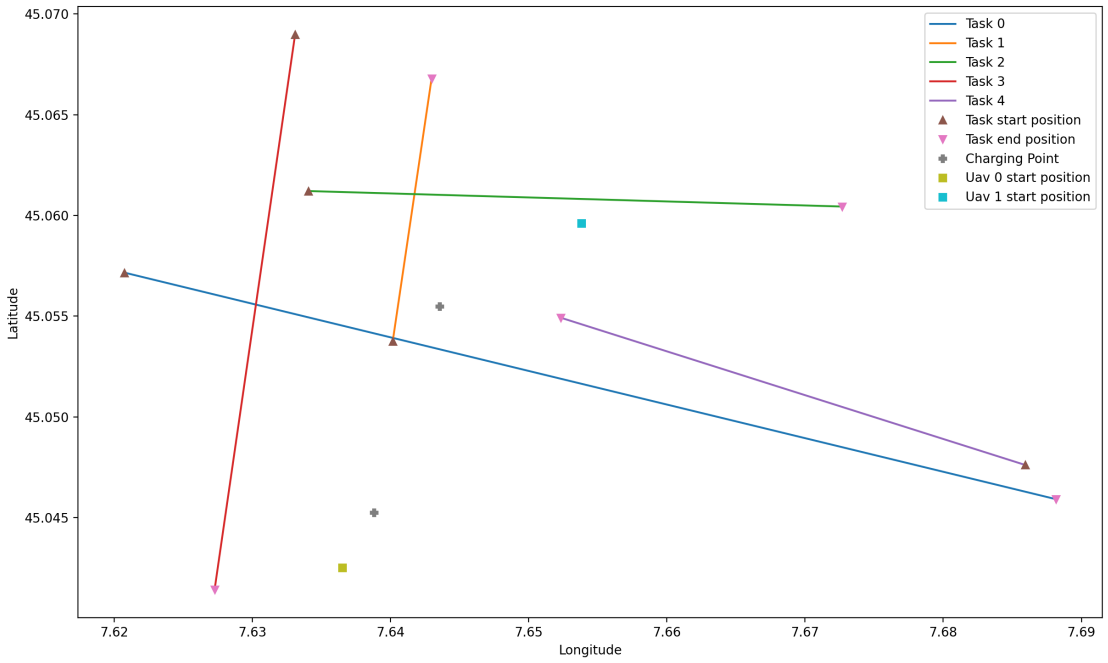

Figure 5.1: Operating environment simple scenario

As a case in point, in Figure 5.3, we can observe the performance of the fitting function during each iteration and how convergence is achieved. It is worth noting that the final solution was obtained after the sixth iteration, but an additional eight iterations were carried out to explore a wider range of possibilities and enhance the likelihood of discovering the local minimum. The significance of this can be grasped by the fact that a minimum was found after iteration 1, which may have appeared to be global, since it was not until the sixth iteration that an improvement was achieved. A higher value of n_{conv} augments the possibility of identifying the global minimum of the fitting function at the cost of increased computational time due to the greater number of iterations. However, this can be circumvented by raising the ϵ tolerance.

At first glance, this allocation may not seem fair, but it is because UAV 0 has a lower payload capacity and therefore consumes less energy for the same distance traveled. This is particularly true given that all tasks are delivered within their time windows. Hence, the algorithm assigns only UAV 1 to task 2, as it exceeds the maximum payload capacity of UAV 0.

From the results of the simple scenario, we can observe that there is no significant difference in the energy consumption between using the *rte* parameter set to `true` or `false`. However, the computation times with `false` are considerably longer.

Version	Run time [s]	J [MJ]	p_{onTime}	Uav 0 tasks	Uav 1 tasks	n_i
V1	5.19	2.03	1.0	1, 4, 3, 5, 0	2	7
V2	4.80	1.85	1.0	1, 0, 4, 6, 3	2	7
V3	1.47	2.53	1.0	3, 5, 0, 4	1, 6, 2	9
V4	1.66	2.03	1.0	1, 4, 3, 5, 0	2	11

Table 5.2: Results of simulations for different algorithm versions in the simple scenario

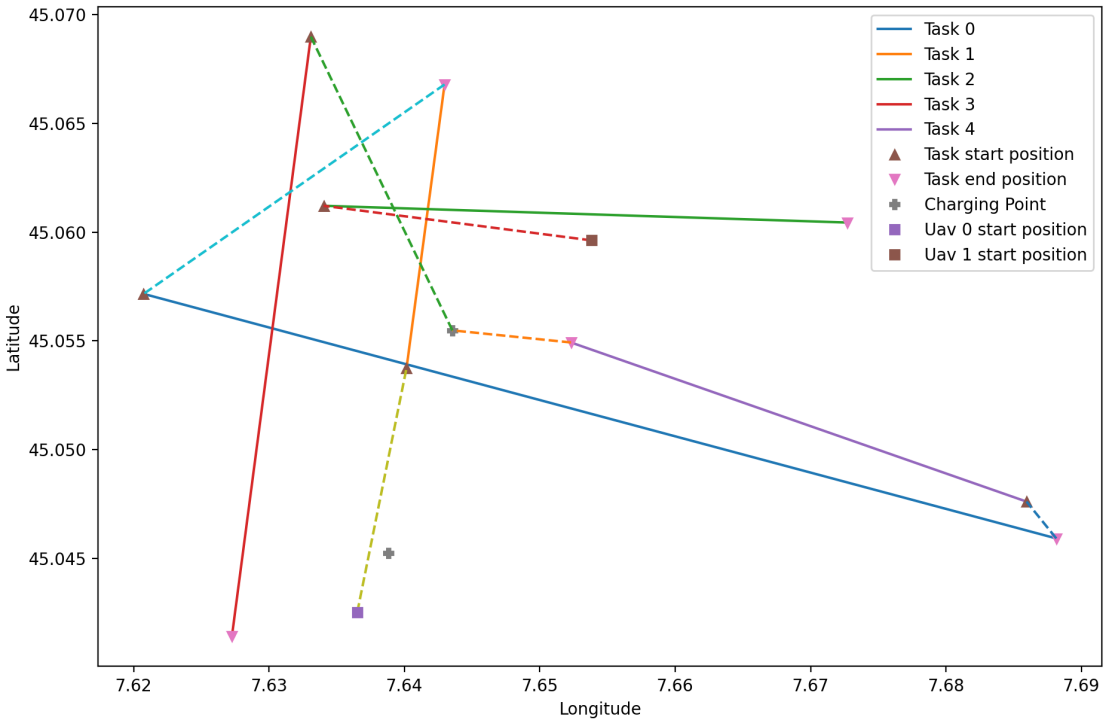

Figure 5.2: Graph solution of V2 task allocation algorithm for the simple scenario

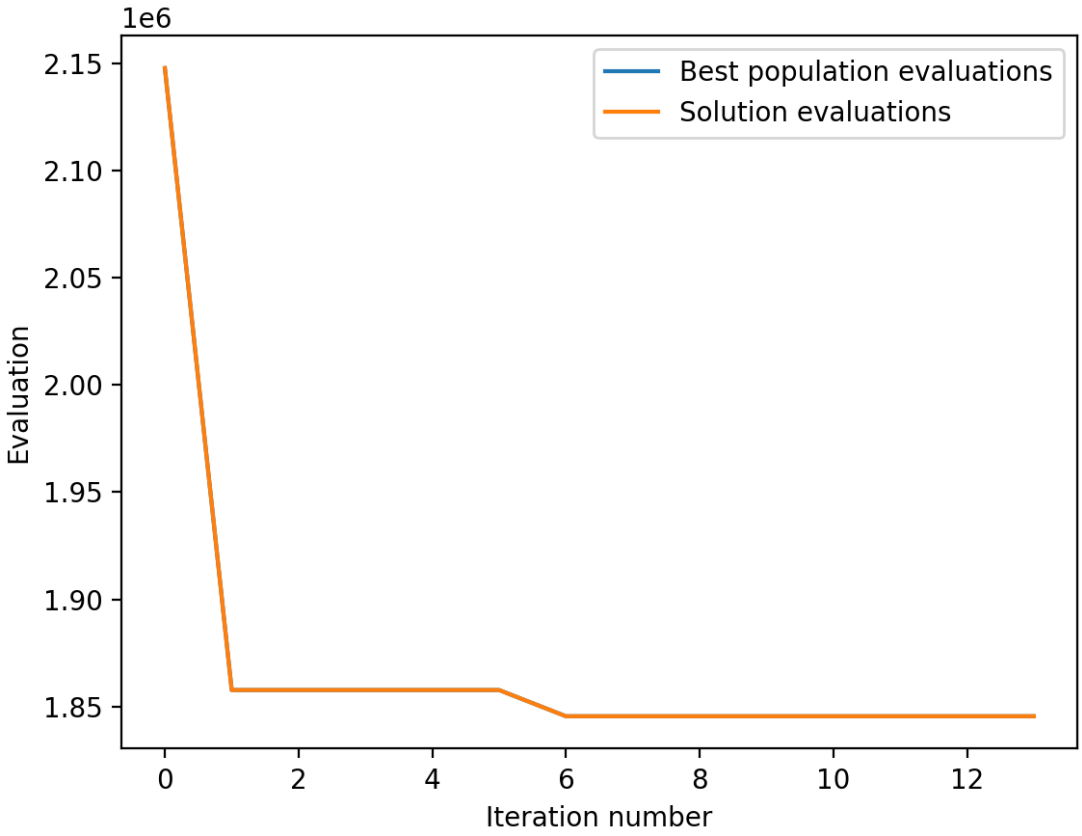

Figure 5.3: Progress of the fitting function for each iteration of the V2 algorithm in the simple scenario

5.2 Complex scenario

In order to evaluate the performance of the genetic task allocation algorithm in complex scenarios, 40 delivery tasks were assigned to 4 UAVs along with 4 charging points on the map. Two scenarios were considered: scenario A, which involves random assignment of the delivery window for each task, and scenario B, which assumes a uniform delivery window of 5 hours for all tasks. Monte Carlo simulations were conducted 20 times for both scenarios using each version of the algorithm. The results are presented in Table 5.3 and Table 5.4, where the mean value μ and standard deviation σ are reported for the total energy required by the fleet to complete all assigned tasks E_{TOT}, the number of iterations done by the algorithm n_i, the total running time of the GA t_{TOT} and the number of charging stations visited n_{char}.

It should be noted that the genetic algorithm assigns all tasks to the entire fleet, and if a solution cannot be found to satisfy the delivery time and payload constraints, the algorithm will return no output. This is why versions 1 and 3 of the algorithm did not produce any results in scenario A. However, versions 2 and 4 achieved a delivery success rate of 86% and 88% respectively, indicating that the minimum energy was achieved by delivering the majority of the parcels within the available delivery window.

Interestingly, version 2 of the algorithm, which adds charging tasks at each cycle rather than just at the end, provided the minimum energy solution. However, the energy savings were only 2.5%, which may not be significant compared to the gain in computation time of 510%.

It is important to note that while the genetic task allocation algorithm showed promising results in these complex scenarios, further research is needed to explore its performance in uncertain and changing environments and to improve its scalability and efficiency.

Similar results can be seen in scenario B, where this time because of the very large time delivery windows for all tasks, we have a solution with all versions of the GA. Comparing the V2 with the V4, the energy is 5.8% lower but the total running time of the algorithm was 6.5 times more.

Version	E_{TOT} [MJ] $\mu \pm \sigma$	p_{onTime} $\mu \pm \sigma$	n_{char} $\mu \pm \sigma$	n_i $\mu \pm \sigma$	t_{TOT} [s] $\mu \pm \sigma$
V1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
V2	15.79 ± 0.52	0.857 ± 0.030	4 ± 0	8.9 ± 4.0	2690.2 ± 1548.8
V3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
V4	16.09 ± 0.60	0.879 ± 0.017	4 ± 0	9.0 ± 5.2	526.7 ± 340.8

Table 5.3: Simulation results complex scenario A

Version	E_{TOT} [MJ] $\mu \pm \sigma$	p_{onTime} $\mu \pm \sigma$	n_{char} $\mu \pm \sigma$	n_i $\mu \pm \sigma$	t_{TOT} [s] $\mu \pm \sigma$
V1	15.06 ± 0.60	1 ± 0	4 ± 0	7.8 ± 3.4	2428.8 ± 1375.3
V2	14.79 ± 0.46	1 ± 0	4 ± 0	8.2 ± 3.2	2354.0 ± 1503.9
V3	15.50 ± 0.62	1 ± 0	4 ± 0	7.4 ± 3.2	347.4 ± 341.2
V4	15.70 ± 0.81	1 ± 0	4 ± 0	8.8 ± 3.6	373.6 ± 199.7

Table 5.4: Simulation results complex scenario B

Chapter 6

Conclusions

The genetic task allocation algorithm proposed in this thesis is based on a genetic algorithm framework, which is a powerful and flexible optimization technique inspired by natural evolution.

The algorithm takes into account various factors, such as task requirements, UAV capabilities, and charging constraints, to generate an optimized task allocation schedule that minimizes energy consumption and computation time. The algorithm also incorporates a novel charging task insertion method to improve the overall efficiency of the schedule.

In terms of results, the algorithm was tested on a variety of problem instances involving different task and UAV configurations. The results demonstrated that the algorithm is effective in generating high-quality task allocation schedules that minimize energy consumption and computation time.

The algorithm's performance was particularly impressive when charging tasks were strategically placed only at the end of the final solution resulted in significant improvements in computation time, without compromising energy efficiency. The algorithm demonstrated exceptional performance under this configuration.

It is important to note that the algorithm does not always guarantee to find the optimal solution, but rather aims to generate a near-optimal solution within a reasonable amount of time.

To achieve even better results, future research could focus on identifying the optimal combination of parameters that would lead to faster convergence to the minimum solution.

While the genetic task allocation algorithm proposed in this thesis has demonstrated its potential, there are some important considerations to keep in mind. For instance, the algorithm assumes perfect information regarding the environ-

ment and task requirements, which may not always be available in practice. Thus, exploring how the algorithm can adapt to uncertain and changing environments is an interesting avenue for future research.

Furthermore, scalability and efficiency are critical factors that should be taken into account when applying the algorithm to larger problem instances. Thus, it is important to investigate how to improve the algorithm's scalability and efficiency without compromising solution quality.

It is important to note that the algorithm does not always guarantee to find the optimal solution, but increasing the number of individuals and iterations, as well as implementing stop conditions for convergence, can improve the probability of finding the minimum solution.

The genetic task allocation algorithm constructed has considerable room for improvement in the view of optimizing it and thus improving computation time. However, it was not the goal of this thesis to make the algorithm more efficient, but rather to show its actual usability in the task allocation scenario for package delivery.

Overall, the genetic task allocation algorithm presented in this thesis provides a promising solution for task allocation problems in the UAV domain. Nevertheless, further improvements and optimizations are necessary to expand its capabilities and applicability in real-world scenarios.

Bibliography

[1] Giuseppe Aiello et al. "Energy consumption model of aerial urban logistic infrastructures". In: *Energies* 14.18 (2021), p. 5998.

[2] Taha Benarbia and Kyandoghere Kyamakya. "A literature review of drone-based package delivery logistics systems and their implementation feasibility". In: *Sustainability* 14.1 (2021), p. 360.

[3] Diego Cattaruzza et al. "A memetic algorithm for the multi trip vehicle routing problem". In: *European Journal of Operational Research* 236.3 (2014), pp. 833–848.

[4] Genshe Chen and Jose B Cruz. "Genetic algorithm for task allocation in UAV cooperative control". In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*. 2003, p. 5582.

[5] Chun Cheng, Yossiri Adulyasak, and Louis-Martin Rousseau. "Drone routing with energy function: Formulation and exact algorithm". In: *Transportation Research Part B: Methodological* 139 (2020), pp. 364–387.

[6] Shushman Choudhury et al. "Efficient large-scale multi-drone delivery using transit networks". In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 757–788.

[7] Konstantinos Dalamagkidis, Kimon P Valavanis, and Les A Piegl. *On integrating unmanned aircraft systems into the national airspace system: issues, challenges, operational restrictions, certification, and recommendations*. Springer, 2009.

[8] Kevin Dorling et al. "Vehicle routing problems for drone delivery". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.1 (2016), pp. 70–85.

[9] Xiaoyu Du et al. "Multi-UAVs cooperative task assignment and path planning scheme". In: *Journal of Physics: Conference Series*. Vol. 1856. 1. IOP Publishing. 2021, p. 012016.

[10] Stefano Giordani, Marin Lujak, and Francesco Martinelli. "A distributed algorithm for the multi-robot task allocation problem". In: *International conference on industrial, engineering and other applications of applied intelligent systems*. Springer. 2010, pp. 721–730.

[11] Sertac Karaman and Emilio Frazzoli. "Incremental sampling-based algorithms for optimal motion planning". In: *Robotics Science and Systems VI* 104.2 (2010).

[12] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. "Distributed algorithms for multirobot task assignment with task deadline constraints". In: *IEEE Transactions on Automation Science and Engineering* 12.3 (2015), pp. 876–888.

[13] Gyeongtaek Oh et al. "Market-based distributed task assignment of multiple unmanned aerial vehicles for cooperative timing mission". In: *Journal of Aircraft* 54.6 (2017), pp. 2298–2310.

[14] Alena Otto et al. "Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey". In: *Networks* 72.4 (2018), pp. 411–458.

[15] Andrew J Page, Thomas M Keane, and Thomas J Naughton. "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system". In: *Journal of parallel and distributed computing* 70.7 (2010), pp. 758–766.

[16] Sabitri Poudel and Sangman Moh. "Task assignment algorithms for unmanned aerial vehicle networks: A comprehensive survey". In: *Vehicular Communications* (2022), p. 100469.

[17] Stefano Primatesta, Alessandro Rizzo, and Anders la Cour-Harbo. "Ground risk map for unmanned aircraft in urban environments". In: *Journal of Intelligent & Robotic Systems* 97 (2020), pp. 489–509.

[18] Stefano Primatesta et al. "An innovative algorithm to estimate risk optimum path for unmanned aerial vehicles in urban environments". In: *Transportation research procedia* 35 (2018), pp. 44–53.

[19] Wei Qin et al. "Multi-agent reinforcement learning-based dynamic task assignment for vehicles in urban transportation system". In: *International Journal of Production Economics* 240 (2021), p. 108251.

[20] Deo Prakash Vidyarthi and Anil Kumar Tripathi. "Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm". In: *Journal of Systems Architecture* 47.6 (2001), pp. 549–554.

[21] Xueli Wu et al. "Multi-UAV task allocation based on improved genetic algorithm". In: *IEEE Access* 9 (2021), pp. 100369–100379.

[22] Anmin Zhu and Simon X Yang. "A neural network approach to dynamic task assignment of multirobots". In: *IEEE transactions on neural networks* 17.5 (2006), pp. 1278–1287.

[23] WANG Zhu et al. "Multi-UAV reconnaissance task allocation for heterogeneous targets using an opposition-based genetic algorithm with double-chromosome encoding". In: *Chinese Journal of Aeronautics* 31.2 (2018), pp. 339–350.