ALGORITHMS FOR MULTI-ROBOT SYSTEMS ON THE COOPERATIVE EXPLORATION & LAST-MILE DELIVERY PROBLEMS

Dissertation written by
Fernando Ropero Pastor

Under the supervision of
Dra. María Dolores Rodríguez Moreno
Dr. Pablo Muñoz Martínez

International advisor
Dr. Erik Steur

Universidad de Alcalá

Dissertation submitted to the Polytechnic School of the University of Alcalá, in partial fulfilment of the requirements for the degree of Doctor of Philosophy

A study on cooperative multi-robot paradigms for the exploration and last-mile delivery problems

Doctoral Degree in Space Research and Astrobiology
Computer Engineering Department
Polytechnic School
University of Alcalá

2020

A mis padres y mi hermana.

ABSTRACT

The emergence of Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) has conducted the research community to face historical complex problems by devising UGV-UAV cooperation paradigms. However, it is usually not a trivial task to determine whether or not a UGV-UAV cooperation is suitable for a particular problem. For this reason, in this thesis, we investigate a particular UGV-UAV cooperation paradigm over two problems in the literature, and we propose an autonomous controller to test it on simulated scenarios.

Driven by the planetary exploration, we formulate a particular cooperative exploration problem consisting of reaching a set of target points in a large-scale exploration area. This problem defines the UGV as a moving charging station to carry the UAV through different locations from where the UAV can reach the target points. Consequently, we propose the cooperaTive ExploRation Routing Algorithm (TERRA) to solve it. This algorithm stands out for splitting up the exploration problem into five sub-problems, in which each sub-problem is solved in a particular stage of the algorithm. In the same way, driven by the explosion of parcels delivery in e-commerce companies, we formulate a generalization of the well-known last-mile delivery problem. This generalization defines the same UGV's and UAV's rol as the exploration problem. That is, the UGV acts as a moving charging station which carries the parcels along several UAVs to deliver them. In this way, we follow the split strategy depicted by TERRA to propose the COoperative Unmanned deliveRIEs planning algoRithm (COURIER). This algorithm replicates the first four TERRA's stages, but it builds a new fifth stage to produce a task plan solving the problem. In order to evaluate the UGV-UAV cooperation paradigm on simulated scenarios, we propose the Autonomous coopeRatIve Execution System (ARIES). This controller follows a hierarchical decentralized leader-follower approach to integrate any cooperation paradigm in a distributed manner.

Both algorithms have been characterized to identify the relevant aspects of the cooperation paradigm in the related problems. Also, both of them demonstrate a great performance of the cooperation paradigm in such problems, and as well as the autonomous controller, reveal a great potential for future real applications.

RESUMEN

La cooperación robótica ha evolucionado desde sistemas estáticos, que intercambian información para completar una tarea en común, a los actuales sistemas dinámicos capaces de coordinarse y adaptarse para resolver un problema en función de los objetivos. Esto se debe en gran medida a la evolución en robótica en las últimas dos décadas. La aparición de sistemas robóticos con diferentes capacidades ha llevado a los investigadores a proponer sistemas cooperativos (también llamados paradigmas de cooperación) que combinen dichas habilidades para afrontar problemas inabordables hace unos años. No obstante, dada la naturaleza compleja de determinados problemas, no resulta trivial conocer el paradigma de cooperación adecuado para resolverlo.

En la literatura se recogen diferentes soluciones cooperativas abordando un amplio espectro de problemas. No obstante, el paradigma de cooperación entre vehículos autónomos terrestres (*Unmanned Ground Vehicles* o UGVs) y vehículos aéreos no tripulados (*Unmanned Aerial Vehicles* o UAVs) es uno de los más estudiados. Esto se debe fundamentalmente a las sinergias existentes entre las capacidades de ambos sistemas. Con el objetivo de profundizar en el conocimiento de este paradigma de cooperación UGV-UAV, esta tesis se centra en el estudio de su comportamiento y rendimiento en dos problemas en particular: la exploración planetaria y la entrega de paquetes.

La exploración planetaria es uno de los problemas donde la cooperación UGV-UAV presenta grandes expectativas. Esta exploración supone un pilar central para las agencias de investigación como la Administración Nacional de la Aeronáutica y del Espacio (*National Aeronautics and Space Administration* o NASA) o la Agencia Espacial Europea (*European Space Agency* o ESA). Un ejemplo de ello es la Mars 2020 Mission, en la que NASA pretende dar los primeros pasos para la exploración cooperativa autónoma. Por este motivo, un objetivo de esta tesis es el estudio de un paradigma de cooperación UGV-UAV en problemas de exploración. Para ello, primero introducimos el problema de exploración llamado *Energy Constrained UAV and Charging Station UGV Routing Problem* (ECU-CSURP). Este problema tiene como objetivo alcanzar una serie de metas en un área de grandes dimensiones, donde se busca seguridad y minimizar la distancia recorrida. Además, los sistemas robóticos no tienen las habilidades ni la energía suficiente para realizar la exploración de forma individual. Por tanto, ambos necesitan cooperar para realizar la exploración satisfactoriamente. Para resolver el problema, proponemos el *cooperaTive ExploRation Routing Algorithm* (TERRA). Este algoritmo implementa un paradigma de cooperación UGV-UAV, donde el UGV es una estación

de carga móvil que transporta al UAV a lugares desde donde el UAV puede cargar su batería y despegar para alcanzar los objetivos de la exploración. Fundamentalmente, TERRA divide el problema de exploración en cinco sub-problemas, y por tanto, cada sub-problema es resuelto en una etapa diferente del algoritmo. Para analizar este algoritmo, hemos evaluado su comportamiento sobre sus parámetros de entrada y la configuración del problema. En este sentido, hemos detectado que su rendimiento depende de la energía disponible de los UAVs y del nivel de agrupación de los objetivos en el área de exploración. No obstante, ECU-CSURP utiliza una representación del terreno poco realista y limitada a entornos planos. Por ello, en esta tesis también extendemos TERRA para poder ejecutarse sobre entornos con elevación. Esta extensión consiste en actualizar las etapas de TERRA a entornos con elevación, e integrar en TERRA un algoritmo que calcula la ruta del UGV considerando parámetros como la pendiente del terreno. En este caso, detectamos que la efectividad del algoritmo depende esencialmente de la complejidad del entorno.

También, el problema de la entrega de paquetes en la última milla siempre ha supuesto un gran reto para las compañías de comercio electrónico. Compañías como Amazon o DHL han anunciado su interés por elaborar una solución cooperativa basada en el paradigma de cooperación UGV-UAV. Desde entonces, la literatura en torno a este problema y el paradigma de cooperación UGV-UAV ha experimentado un crecimiento exponencial. Por este motivo, otro objetivo de esta tesis es el estudio del mismo paradigma de cooperación UGV-UAV en el problema de la entrega de paquetes en la última milla. Para ello, primero formulamos el problema de entrega de paquetes llamado *multiple UAVs and Charging station Vehicle Last-Mile delivery Problem* (mUCVLMP). Este problema tiene como objetivo entregar un conjunto de paquetes en diferentes localizaciones distribuidas en un área de grandes dimensiones en el menor tiempo posible. Además, estos paquetes han de ser entregados por los UAVs. No obstante, los UAVs no tienen energía suficiente para realizar todas las entregas por si mismos, por lo que requieren de un UGV que les transporte. Para resolver el problema, proponemos el *COoperative Unmanned deliveRIEs planning algoRithm* (COURIER). Este algoritmo implementa el mismo paradigma de cooperación UGV-UAV implementado por TERRA para problemas de exploración. Además, en este caso, el algoritmo permite a varios UAVs realizar entregas simultáneas. De este modo, COURIER se construye con las primeras cuatro etapas de TERRA, pero a diferencia de éste, implementa una nueva quinta etapa que calcula el plan de entregas completo para resolver el problema. El análisis de este algoritmo nos muestra que la energía disponible en los UAVs no es un factor tan determinante como pensábamos, si no que la velocidad de estos juega un papel importante también. Además, el aumento de

UAVs disponibles no es siempre sinónimo de minimizar los tiempos de entrega.

Finalmente, en esta tesis proponemos también un controlador autónomo para llevar a cabo la ejecución y testeo del paradigma de cooperación UGV-UAV utilizado para resolver los problemas de exploración y entrega de paquetes. El controlador llamado *Autonomous coopeRative Execution System* (ARIES), comprende una arquitectura de control que permite el despliegue de cualquier modelo de cooperación entre diferentes sistemas robóticos. ARIES es una arquitectura descentralizada construida sobre el sistema de ejecución *Teleo-Reactive EXecutive* (T-REX). Además, utiliza un esquema jerárquico en el que hay un líder y seguidores, donde el líder es el encargado de deliberar los planes cooperativos y coordinar la ejecución de los seguidores. En cambio, los seguidores se limitan a ejecutar las acciones encomendadas por el líder. Para la evaluación de la arquitectura hemos utilizado escenarios simulados.

"We have to light up the darkness"

— Robert N. Marley, 1945

ACKNOWLEDGMENTS

En primer lugar, quiero expresar mi más sincero agradecimiento, respeto y admiración hacia mis directores de tesis, María Dolores Rodríguez Moreno y Pablo Muñoz Martínez. Desde el primer día hasta el último, me habéis ayudado desinteresadamente a construir los cimientos de un camino profesional que desconocía, pero que ahora me guía. Desde el inicio habéis depositado una confianza ciega en mí, y me habéis prestado toda la atención y ayuda a vuestro alcance. Ojalá pudiese expresar todas las gracias que os merecéis.

También, quiero expresar mis agradecimientos a la familia del grupo de sistemas inteligentes de la Universidad de Alcalá. Sobre todo a David Fernández Barrero, pues eres un auténtico oráculo de conocimiento donde acudir, y espejo de ética y trabajo donde reflejarse. Gracias por tus infinitos consejos que también me has ofrecido siempre de forma desinteresada. Agradecer también a todos los compañeros de laboratorio que forman la familia: Dani, Armando y Mario; cuánto se aprende estando rodeado de gente como vosotros todos los días.

Por supuesto, gracias a mis padres Pedro y Encarna, y mi hermana Encar, pues son la razón de ser quien soy y de haber alcanzado la meta profesional que he alcanzado. Gracias por enseñarme la honestidad, bondad y esfuerzo que hoy me permiten afrontar cualquier reto en mi vida. Cada día, me enseñáis valores que hoy llevo por bandera.

No me olvido de mis amigos, mi familia elegida, que han aguantado alguna que otra charla distendida y aburrida sobre mi trabajo durante los últimos años. Ha sido un placer daros la chapa. Y siempre, siempre, quiero dar las gracias a mis amigos de siempre, los que están y los que no, porque siempre han estado y estarán ahí. Siempre resonará tu eco.

During my research stay in Delft, The Netherlands, I spent a wonderful time with a lot of researchers working in the Delft Center for Systems and Control at the Delft University of Technology. In particular, I would like to express my gratitude and respect to the professor that allowed me spend such a wonderful time, the Dr. Erik Steur. Many thanks Erik for your advices and support during my stay.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
NASA	National Aeronautics and Space Administration
ESA	European Space Agency
MHS	Mars Helicopter Scout
ILP	Integer Linear Programming
TSP	Travelling Salesman Problem
VLSI	Very Large Scale Integration
WSN	Wireless Sensor Network
AUV	Autonomous Underwater Vehicle
mTSP	Multiple Travelling Salesman Problem
TSPPD	Travelling Salesman Problem with Pickup and Delivery
NNS	Nearest Neighbour Search
SCP	Set Cover Problem
HSP	Hitting Set Problem
aTSP	asymmetric Travelling Salesman Problem
sTSP	symmetric Travelling Salesman Problem
mTSPTW	multiple Travelling Salesman Problem with Time Windows
VRP	Vehicle Routing Problem
DVRP	Distance-constrained Vehicle Routing Problem
ASV	Autonomous Surface Vehicle
ISR	Intelligence, Surveillance and Reconnaissance
GPS	Global Positioning System
FSTSP	Flying Sidekick Salesman Problem
TSP-D	Travelling Salesman Problem with Drone
TSP-mD	Travelling Salesman Problem with multiple Drones
MILP	Mixed Integer Linear Programming
mFSTSP	multiple Flying Sidekick Salesman Problem
AI	Artificial Intelligence
ECSS	European Cooperation for Space Standardization
RAP	Reactive Action Package
ATLANTIS	A Three-Layered Architecture for Navigating Through Intricate Situations

LAAS	Laboratory of Analysis and Architecture of Systems
CLARAty	Couple Layered Architecture for Robotic Autonomy
IDEA	Intelligent Distributed Execution Architecture
TREX	Teleo-Reactive EXecutive
ASyMTRe	Automated Synthesis of Multi-robot Task solutions through software Reconfiguration
HiDDeN	High-level Distributed DecisioN
ECU-CSURP	Energy Constrained UAV and Charging Station UGV Routing Problem
TERRA	cooperaTive ExploRation Routing Algorithm
DTM	Digital Terrain Model
HiRISE	High Resolution Imaging Science Experiment
mUCVLMP	multiple UAVs and Charging station Vehicle Last-Mile delivery Problem
COURIER	COoperative Unmanned deliveRIEs planning algoRithm
MA	Memetic Algorithm
LRLS	Low-Range & Low-Speed
LRHS	Low-Range & High-Speed
HRLS	High-Range & Low-Speed
HRHS	High-Range & How-Speed
T-REX	Teleo-Reactive EXecutive
ARIES	Autonomous coopeRatIve Execution System
PDDL	Planning Domain Definition Language
NDDL	New Domain Definition Language
DDL	Domain Definition Language
GER	Generic Executive Reactor
UP2TA	Unified Path Planning and Task Planning Architecture
R-Reactor	Recover-Reactor
V-REP	Virtual Robot Experimentation Platform

Part I

THE FOUNDATIONS

The human being is relentlessly bound to its natural evolution in an unknown and dynamic world. Its evolution implies bringing out new knowledge aiming to acquire natural skills to adapt and progress in its habitat. The human evolution and social history tell us that mostly this learning is cultivated thanks to the exploration and observation of the environment. The space research, and more precisely the planetary exploration, raised under this fundamental pillar. Since its foundations, the space industry has invested a lot of resources to explore hostile and unknown environments which venture our world knowledge into new dimensions.

1 INTRODUCTION

In this chapter we present the basis of the work done in this PhD. First, we describe the motivation that conducts our research. Then, we define the objectives and structure of this dissertation. Finally, we enumerate the publications related to this PhD.

1.1 MOTIVATION

As in the human evolution, the cooperation among robotic systems enables to tackle complex problems that individually might be too intricate. In fact, robotics cooperation is not a new concept, but it always has been part of robotics. However, it could not evolve properly due to technology limitations at that time. Nowadays, years of technological growth have placed robotics cooperation in the spotlight of the international research community. Thus, if we follow the cooperation role in the human evolution, we can envision a bright future for the robotics cooperation.

Space agencies, such as the National Aeronautics and Space Administration (NASA) or the European Space Agency (ESA), are defining new cooperation paradigms aiming to improve the quality and quantity of the scientific return from their exploration missions. One of the most studied problems is the area coverage using multiple Unmanned Ground Vehicles (UGVs) [1] or Unmanned Aerial Vehicles (UAVs) [2]. Also, we can highlight NASA's plans to launch the Mars Helicopter Scout (MHS) along with the Mars 2020 Rover [3]. However, the MHS is an experimental technology with which NASA pretends to demonstrate that controlled flight can be executed in the extremely thin Martian atmosphere. In this way, the MHS could serve as scout to the rover or human pioneers by taking aerial pictures. In spite of this cooperation will be managed by a ground operator, the Mars 2020 mission represents a first approach for UGV-UAV cooperation. Besides, NASA's objective is to deploy heterogeneous (robots with different capabilities) UGV-UAV systems that are capable of performing cooperative missions. These first approaches on cooperation paradigms in the space industry ignited our motivation and driven us for bringing out a research line to which the present PhD work belongs to.

In line with the robotics cooperation, several e-commerce companies around the world, such as Amazon, FedEx, DHL or UPS, are investing a lot of resources to reduce delivery times and expenses. This is the well-known last-mile delivery problem in the literature and, a lot of research lines [4–6] have began since the logistics companies decided over 2015 to introduce multiple UGV-UAV cooperation systems into their delivery services. Since then, the research community is focused on novel delivery services involving robotic systems which can deploy a high level of autonomous cooperation to tackle the problem from different angles. For instance, one of them uses UAVs to depart from the warehouse (or a human-driven truck) and travel to the customer location for parcel delivery. Once the parcel is delivered, the UAVs return to the warehouse. Some companies only use UAVs for the parcel delivery, such as Amazon, FedEx or DHL, but others, such as UPS, deploys a human-driven truck to help the UAVs reach their destinations. Nowadays, the state of the art in robotics cooperation for delivery services reveals a study field in effervescence. The similarities between the cooperation paradigms to solve the exploration and the last-mile delivery problems, along with the current explosion of deliveries of e-commerce companies, guided our research work through package delivery problems.

This PhD aims to bring the UGV-UAV cooperation paradigm as a feasible solution in the robotics industry. Our work is focused on studying a particular UGV-UAV cooperation paradigm in the exploration and the package delivery problems. The paradigm we study uses the UGV to carry the UAV through different locations from where the UAV can reach locations and return to the UGV after completing the assigned tasks. For addressing each problem, we will design and build one algorithm. Also, we will devise an autonomous controller to test the paradigm in simulated scenarios. The state of the art about this cooperation paradigm just started a few years ago, but we consider that new studies are required to better investigate the performance in these particular scenarios.

1.2 OBJECTIVES

This PhD focuses on the study of the UGV-UAV cooperation paradigm on the exploration and package delivery problems. Also, we want to demonstrate the effectiveness of the UGV-UAV cooperation paradigm in simulated scenarios. Therefore, the objectives of the present research work can be summarized as follows:

1. Formulating an exploration problem to be addressed by the UGV-UAV cooperation paradigm.
2. Creating a two-dimensional path planning algorithm considering the mathematical constraints and objectives of the exploration problem.
3. Extending the previous algorithm to a three-dimensional path planning algorithm. The goal is to also consider terrain features to extract feasible paths and so, to generate cooperative three-dimensional paths in real scenarios.

4. Formulating a generalization of the last-mile delivery problem to be addressed by the UGV-UAV cooperation paradigm.

5. Creating a task planning algorithm to solve the formulated generalization of the last-mile delivery problem. Here, the idea is to extend the previous two-dimensional path planning algorithm by considering new constraints into the original problem, e. g., temporal coordination or multiple UAVs.

6. Conducting experiments to analyse the performance of the path planning and task planning algorithms comparing with other state of the art algorithms.

7. Implementing an autonomous controller capable of managing a multi-robot system following the UGV-UAV cooperation paradigm. The objective is to build a cooperative controller using some existing technology in the literature.

8. Performing simulated experiments to evaluate the behaviour of the autonomous controller, and so, laying the foundations of a robotics controller able to deploy the UGV-UAV cooperation paradigm studied in this research work.

1.3 STRUCTURE AND CONTENTS

The structure of this dissertation can be outlined in the following six chapters:

- Chapter 1: describes the motivation, objectives, structure and resulting publications of this thesis.

- Chapter 2: presents the state of the art for the different fields that have been covered. First, it introduces the basis of the discrete optimization theory applied in the robotics engineering field. Second, it summarizes particular approaches where cooperative multi-robot systems are deployed to overcome different problems. Last, it describes the evolution of autonomous controllers designed for multi-robot cooperation.

- Chapter 3: proposes a path planning algorithm to exploit a UGV-UAV cooperation paradigm on exploration problems. First, it describes a mathematical formulation which contains the definition and constraints of the exploration problem. Then, it presents a two-dimensional path planning algorithm integrating the problem definition, and a three-dimensional path planning considering the terrain features. The two-dimensional algorithm has been evaluated using randomly generated maps, meanwhile the three-dimensional algorithm uses real Mars maps.

- Chapter 4: proposes a task planning algorithm to solve a generalization of the last-mile delivery problem. First, it presents the mathematical formulation integrating new constraints to the original problem definition and, second, the task planning algorithm modelled as an evolutionary algorithm. The algorithm has been evaluated using the same random map generator used for the previous algorithm.

- Chapter 5: describes an autonomous controller for the UGV-UAV cooperation paradigm exploited in the previous algorithms. It defines the architecture and its functionality. The controller has been evaluated under simulated environments.

- Chapter 6: presents the conclusions and discusses some relevant future research directions.

1.4 PUBLICATIONS

The results of this PhD have produced several publications in the field of AI in multi-robot cooperation and autonomous controllers. The list of publications is presented as follows:

Fernando Ropero, Daniel Vaquerizo, Pablo Muñoz, and María D. R-Moreno. "An Advanced Teleassistance System to Improve Life Quality in the Elderly". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. (2017), pp. 533–542.

Fernando Ropero, Pablo Muñoz, María D. R-Moreno, and David F. Barrero. "A Virtual Reality Mission Planner for Mars Rovers". In: *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. IEEE. (2017), pp. 142–146.

Pablo Muñoz, María D. R-Moreno, David F. Barrero, and Fernando Ropero. "MoBAr: a Hierarchical Action-Oriented Autonomous Control Architecture". In: *Journal of Intelligent & Robotic Systems* (2018), pp. 1–16.

Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. "A Strategical Path Planner for UGV-UAV Cooperation on Mars Terrains". In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. (2018), pp. 106–118.

Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. "A Versatile Executive Based on T-REX for Any Robotic Domain". In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. (2018), pp. 79–91.

Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. "TERRA: A path planning algorithm for cooperative UGV–UAV exploration". In: *Engineering Applications of Artificial Intelligence* 78 (2019), pp. 260–272.

Fernando Ropero, Daniel Vaquerizo-Hdez, Pablo Muñoz, David F. Barrero, and Maria D. R-Moreno. “LARES: An AI-based teleassistance system for emergency home monitoring”. In: *Cognitive Systems Research* 56 (2019), pp. 213–222.

Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. “ARIES: An Autonomous Controller For Multirobot Cooperation”. In: *IEEE Aerospace and Electronic Systems Magazine* 34.3 (2019), pp. 40–55.

2 STATE OF THE ART

This chapter contains the state of the art of our research work. First, we introduce the basis and nomenclature of classic optimization problems and algorithms applied to robotics. Then, we focus on particular heterogeneous cooperative UGV-UAV(s) approaches for different robotic applications. Finally, we describe the historical evolution and taxonomy in autonomous controllers for multi-robot cooperation.

2.1 DISCRETE OPTIMIZATION IN ROBOTICS: PROBLEMS & ALGORITHMS

The discrete optimization theory is a branch of mathematical optimization which studies problems whose variables can only take discrete values. For instance, some decisions we make in our life are discrete decisions, that is, particular choices that help us to overcome our daily problems, such as hang out with friends versus writing a thesis. A lower number of options helps us to select the particular choices that solve straightforward problems. However, if the problem requires making a sequence of decisions, the dimensionality of these problems increases in an explosion of possible combinations. Simple problems can become exponentially hard with just a few possibilities. Because of this, the study of discrete optimization has always been attractive for the research community since the 18th century.

Many discrete optimization problems in the literature can be formulated as Integer Linear Programming (ILP), where the variables are restricted to be integers and the objective function and constraints are linear. The formula can be expressed in the form

$$\begin{aligned} \text{min (or max)} \quad & \sum_{j=1}^{n} c_j \, x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \forall i \in \{1, 2, \ldots, m\} \\ & \quad x_j \geq 0 \quad \forall j \in \{1, 2, \ldots, n\} \end{aligned} \tag{2.1}$$

Here, the x_j are the decision variables constrained by non-negativity and the coefficients a_{ij}, b_i, c_j are given rational numbers in m inequalities. The goal is to minimize (or maximize) the sum of x_j given weights c_j.

Since the technological advances introduced the computers in the 19th century, there have appeared discrete problems which have

Table 2.1: Combinatorial optimization problems and their robotic applications

Problem name	Complexity	Robotic applications
Set Packing	NP-Complete	VLSI design [7–9] Logistics [10–12] WSN [13–15]
Set Covering	NP-Complete	Vehicle routing [16, 17] Task Allocation [18, 19]
Knapsack	NP-Complete	Scheduling [20–23] Task Allocation [24, 25] WSN [26, 27]
Generalized assignment	NP-Complete	Task Allocation [28–30] Power Management [31, 32]
Travelling salesman	NP-Complete	Vehicle routing [33, 34] Scheduling [35–37] Pick up and delivery [38]

helped to progress a broad spectre of research fields in robotics engineering. Over time, the great variety of problems induced the creation of a taxonomy in different categories. From the robotics engineering perspective, we can mention two relevant categories of discrete problems: combinatorial optimization problems and visibility & proximity problems.

Combinatorial optimization problems are meant to find an optimal object from a finite set of objects. This finite set of objects is a discrete set or can be reduced to discrete. Table 2.1 contains a brief list with combinatorial optimization problems applied in robotics. Each row contains a different problem, its complexity type (which we will explain later), and some relevant robotic applications tackling the problem. As can be noted, the task allocation and scheduling applications are the most common topics. The task allocation is addressed through the Set Covering [18, 19], Knapsack [24, 25] and Generalized assignment [28–30] problems, whereas scheduling applications are tackled through the Set Cover Problem (SCP) [16, 17] and the Travelling Salesman Problem (TSP) [33, 34]. These topics investigate the problems of achieving an optimal task assignment to a multi-robot system subject to different constraints, such as deadlines or energy constraints. Also, the vehicle routing topic is one of the most studied problems in the robotics literature thanks to the TSP. Due to the rising of new robotics systems such as the UAVs or AUVs, the research community

Table 2.2: Proximity and visibility optimization problems and their robotic applications.

Problem name	Complexity	Robotic applications
Nearest neighbour search	$\mathcal{O}(\log N)$	Motion planning [41–43] Sensor placement [44, 45]
Art gallery	NP-Complete	Sensor placement [46] Surveillance [47–49]
Facility location	NP-Complete	Sensor placement [50, 51] Coverage control [52–54]
Euclidean min. spanning tree	$\mathcal{O}(N \log N)$	Task planning [55, 56] Vehicle Routing [57–59]

is devoting efforts in different generalizations of the TSP to create new multi-robot paradigms. For instance, the Multiple Travelling Salesman Problem (mTSP) [34] which consist of finding tours for all salesman, who all start and end at the depot, such that each intermediate node is visited exactly once and the total cost of visiting all nodes is minimized. Other example is the Travelling Salesman Problem with Pickup and Delivery (TSPPD) [39] which consists of determining a minimum cost tour such that each pickup vertex is visited before its corresponding delivery vertex.

Proximity & visibility optimization problems are those which can be stated in terms of geometry. This spectrum of problems and the study of their solutions from a geometrical perspective was originally initiated by Preparata and Shamos [40]. The proximity term means distances estimation between geometric objects, and the visibility term is a mathematical abstraction of the real-life notion of visibility. These problems have been subject matter in several topics related to the robotics engineering field, such as motion planning or task planning. Table 2.2 shows a brief list of very-well studied problems and some applications in the robotics engineering field. Each row shows a particular problem, its complexity type, and some relevant robotic applications tackling the problem. The reader can appreciate that these problems are particularly focused on smart coverage topics, such as surveillance or sensors placement, than the combinatorial optimization problems. This is because of their inherent geometrical properties, which makes them affordable using geometric algorithms.

In addition to the rising of computers in the 19th century, the intrinsic significance of the hardness bounded to these problems, encouraged a complete theory to study their taxonomy from their difficulty measured in time terms[1]: the complexity theory. This theory classifies the problems according to their inherent difficulty. The P class[2] [61] classifies the problems which can be solved by a deterministic Turing machine in polynomial time. The NP class[3] classifies the problems by stating that: a problem p is in NP if every solution to every problem instance of p can be verified in polynomial time. There have been strong efforts in the literature to demonstrate that problems in NP are tractable. A significant study around this research line has been the NP-completeness theory introduced by Cook [62, 63]. This theory states that NP-Complete problems are the hardest problems in NP. Also, Cook [62] defined the first NP-Complete problem, the satisfiability problem, and proved that a problem is NP-Complete if it is in NP and can be reduced to the satisfiability problem. Therefore, if there exist a polynomial time algorithm for even one of the NP-Complete problems, there is at least one polynomial time algorithm for all the problems in NP. A year later, Karp [64] provided a list of 21 NP-complete problems (Karp 21's list)[4] proving that these problems are as least as hard as the satisfiability problem. Since then, the study of NP-Complete problems has been a key topic in the research community.

A problem is tractable if it can be solved by polynomial time algorithms

Thus, as there are not polynomial time optimization algorithms to solve NP-Complete problems (unless $P = NP$), it is required to find a lower factor polynomial time algorithm to solve them. Different methods [65] can be applied to implement algorithms that find a sub-optimal solution in polynomial time to the NP-Complete problems: approximation, randomization or heuristics methods are the most studied. Let us recommend to the reader the book of Cormen et al. [66], which rigorously covers a broad range of algorithms in depth.

Probably the oldest method to design lower factor polynomial time algorithms is the approximation method. This method looks for how closely it is possible to approximate optimal solutions to such problems in polynomial time [67–70]. For instance, we can highlight the work of Vazirani [71], which contains a comprehensive compendium of approximation algorithms for combinatorial and proximity & visibility (called as geometric) problems. Following this method, there are several techniques to design approximation algorithms such as greedy algorithms [72, 73], local search [74, 75], dynamic programming [76, 77] or linear programming [78, 79].

A well-studied method is the randomization method, which introduces a degree of randomness in its functional logic aiming to be faster in running time and simpler to handle than their best known deter-

1 There exist also a classification in space terms, which is the memory required by an algorithm to execute a program and compute a solution [60].
2 P = deterministic Polynomial time.
3 NP = Non-deterministic Polynomial time.
4 The set packing, set covering and knapsack problems in Table 2.1 are in the Karp 21's problems list.

ministic counterparts. Due to their simplicity and speed, randomized algorithms have been an important research subject in optimization problems [80]. Historically, the first randomized algorithms where oriented to solve geometric problems. The Rabin's algorithm [81] is known for being the oldest randomized algorithm. The work of Agarwal and Sharir [82] and Smid [83] contain several randomized algorithms applied to geometric problems. However, a meta-heuristic optimization technique, which develops evolutionary algorithms, has gained reputation in the randomized computation over the last two decades. This technique aims to solve optimization problems inspired by the biological evolution. There are different evolutionary algorithms primarily suited for optimization problems, but the majority derive from genetic algorithms [84–86] or swarm algorithms [87, 88]. For instance, we can mention the Ant colony optimization algorithm designed by Colorni et al. [89], a swarm algorithm based on the pheromone communication method of ants to find the optimal path. This method is frequently used to solve routing problems [90].

The heuristics methods are also deeply studied in the existing literature. A heuristic algorithm implements a clear strategy of finding a sub-optimal solution in the set of all feasible solutions, and thus, solving an optimization problem more quickly than other classic search algorithms. Thus, the main questions related to heuristics algorithms used to be: how do you find a good heuristic? Or how do you evaluate its effectiveness?. The work of Pearl [91] deals with these questions in a rigorous manner, and presents basic heuristic-search procedures used in the literature to solve a broad number of optimization problems. In this way, the heuristics methods have been relevant in the robotics engineering field [92]. They have contributed to get sub-optimal solutions to many robotic problems (see Tables 2.1 and 2.2) in a feasible time. Some of the most studied problems are: motion and routing problems [93–95], coverage control problems [16, 96] or task planning problems [97–99].

As the reader can appreciate, the existing literature studying the discrete optimization problems and algorithms is overwhelming. For that reason, together with the goal of this dissertation is far away of providing a wide survey about discrete optimization, the next subsections describe the particular problems and algorithms that have been under study in this PhD.

2.1.1 *Nearest Neighbour Search*

The Nearest Neighbour Search (NNS) is a form of proximity search and it consists in finding the point in a given set that is nearest to a given point. Formally, Knuth [100] formulates it as follows:

Problem 2.1. Given N points in the plane, with preprocessing allowed, how quickly can a nearest neighbour of a new given query point q be found?.

Here, the nearest neighbour concept [40] is a relation between two points on a set such that: point b is a nearest neighbour of point a, denoted $a \to b$, if

$$\text{dist}(a, b) = \min_{c \in S - a} \text{dist}(a, c) \tag{2.2}$$

The current state of the art about the NNS reflects a deep study in approximation algorithms based on space partitioning. A widely used algorithm for the NNS is the kd-tree method proposed by Friedman et al. [101]. Arya et al. [102] present an optimal algorithm to solve the NNS in fixed dimensions, demonstrating that $(1 + \varepsilon)$-approximations to the k nearest neighbours of q can be computed in $\mathcal{O}(kd \log N)$ time, where d is a real d-dimensional space and ε is any positive real number. Fukunage and Narendra [103] describe a branch-and-bound algorithm eliminating the computing of needless distances which makes it faster. Inspired by computer visions applications, Muja and Lowe [104] propose a system to help the operator to determine the most appropriate algorithm given a particular dataset, and describe a new search algorithm based on hierarchical k-means trees. However, there are studies stating that the performance of space partitioning algorithms generally degrades as dimensionality increases. Weber et al. [105] provide a detailed analysis about the previous affirmation and proposes the sequential *VA-file* scheme outperforming space partitioning algorithms. Also, there are other recent algorithms with promising results, such as the locality-sensitive hashing [106, 107] based on randomized algorithms or the priority search k-means tree [108].

Nevertheless, as we explain in the above section, the intrinsic nature of this problem classifies it as a proximity problem. This made Preparata and Shamos [40] to address the problem from a geometric perspective. In their work, Preparata and Shamos [40] formulated the loci of proximity problem to solve the NNS, which states as follows: Given a set S of N points in the plane, for each point p_i in S, what is the locus of points (x, y) in the plane that are closer to p_i than to any other point of S?. As the solution of this problem is a partition of the plane into regions, they noted that just searching into this space of regions, they could solve the NNS. Thus, Preparata and Shamos [40] were the first who proposed the Voronoi diagrams [109] to solve the NNS problem.

The research work presented in this dissertation uses the notation of Preparata and Shamos [40] for Voronoi diagrams. This notation formulates a Voronoi diagram as follows:

$$V(i) = \bigcap_{i \neq j} H(p_i, p_j) \tag{2.3}$$

where $V(i)$ is the Voronoi polygon associated to p_i and it is formed by the intersections of the half-planes $H(p_i, p_j)$. So, $H(p_i, p_j)$ contains the set of points closer to p_i than to p_j, and it is defined by the perpendicular bisector of $\overline{p_i p_j}$. The convex net of Voronoi polygons form the called Voronoi diagram (see Figure 2.1). Preparata and Shamos [40] demonstrate that, computing the Voronoi diagram of a set of N points, the NNS can be performed in $\mathcal{O}(\log N)$ time, using $\mathcal{O}(N)$ storage and $\mathcal{O}(N \log N)$ preprocessing time, which is optimal.

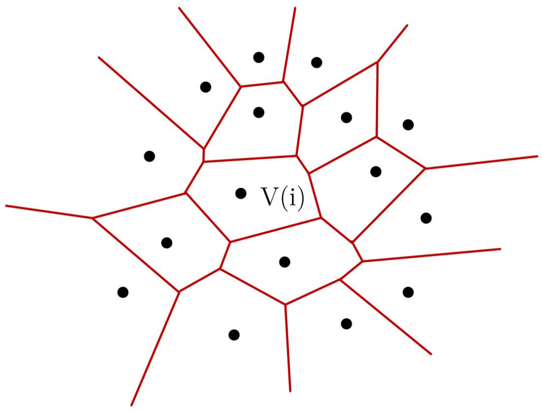


Figure 2.1: Voronoi diagram formed by the convex net of Voronoi polygons of every point.

Voronoi diagrams have been commonly used in the robotics engineering field due to its geometric properties in space partitioning. One of the most common applications is path planning [93, 110], where are used to partition the terrain in Voronoi polygons to obtain smoothed optimal paths avoiding obstacles and terrain features. Other application are localization problems [111] to estimate the robot's position in the terrain or to correct robot's odometry. Also, they have been applied in coordinated multi-robot exploration [112–114] for space partitioning of the occupancy cells among the robots.

2.1.2 *Set Cover Problem*

The SCP is one of the Karp's 21 NP-Complete problems. As we mentioned above, Karp published a paper [64] where he demonstrated that twenty one combinatorial problems are a reduction from the boolean satisfiability problem enunciated by Cook [62] (see section 2.1), and

Algorithm 1 Greedy set cover algorithm

```
$C \leftarrow \emptyset$
while $C \neq U$ do
    Get the minimum cost-effective set $s$ in $S$
    Update the cost-effectiveness of $s$ as: $f = \frac{cost(s)}{S-C}$
    for $e \in S - C$ do
        Update cost of picking $e$ equal to $f$
    end for
    $C \leftarrow C \cup s$
end while
return $C$
```

thus, showing these problems were NP-Complete. Then, Vazirani [71] formulates the SCP as follows:

Problem 2.2. Given a universe U of n elements, a collection of subsets $S = \{s_1, ..., s_k\}$, and a cost function $c : S \leftarrow Q^+$, find a minimum cost subcollection of S that covers all elements of U.

Additionally to the notation, Vazirani [71] provides an ILP formulation of the problem, where x_s is the decision variable which will take 1 if set S is picked in the set cover, and 0 otherwise:

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in S} c_s x_s \\ \text{subject to} \quad & \sum_{s: e \in S} x_s \geq 1 \qquad \forall e \in U \\ & x_s \in \{0,1\} \quad \forall s \in S \end{aligned} \tag{2.4}$$

There are plenty of algorithms solving the SCP and its generalizations. Perhaps, the first approximation algorithm designed to solve the SCP is the greedy heuristic depicted by Johnson [67], Chvatal [115], and Lovász [116] in 1979. As Algorithm 1 shows, this greedy heuristic iteratively picks the most cost-effective set until all elements in the universe are covered. From this work, other approximation algorithms came after, such as the rounding technique proposed by Hochbaum [117], which achieves an approximation factor of the frequency of the most frequent element by using linear programming relaxation [71]. Also, there are other algorithms referred to generalizations of the SCP, such as the vertex cover [118], geometric set cover [119], weighted vertex cover [120], Hitting Set Problem (HSP) [121] or exact cover [122].

2.1.3 *Travelling Salesman Problem*

The TSP is probably the most studied optimization combinatorial problem in the literature. It is a generalization of the Hamiltonian cycle problem, which gets its name from the Irish mathematician W.

R. Hamilton, who invented the icosian game during the 18th century [123]. This game is about finding a Hamiltonian cycle in the edge graph of a dodecahedron. A Hamiltonian cycle is an undirected or directed graph that visits each vertex exactly once. In 1972, as well as the SCP previously described, Karp proved that the Hamiltonian cycle problem is a NP-Complete problem [64], which implied the NP-hardness of the TSP.

The article *A brief History of the Travelling Salesman Problem* published by Cummings [124], states that the first reference of the TSP was reported by Menger (as the messenger problem) in 1932 [125]. At the time, the TSP was defined as: the task of finding, for a finite number of points whose pairwise distances are known, the shortest path connecting the points. Also, Menger proved the rule that: going from the starting point to its nearest point, it does not in general result in the shortest path. Since then, we can tell that the TSP has had a meteoric rise in several research fields on the literature. From a mathematical perspective, the TSP can be formulated as follows:

Problem 2.3. Given an undirected graph $G = (V, E)$ and non-negative distances $d : E \to \mathbb{Z}^+$ on the edges, the goal is to find a tour $T = \{t_i, \dots, t_n\}$ for $i = 1, \dots, n$ that visits every $v_j \in V$ in G for $j = 1, \dots, n$ exactly once, where $t_i = t_n$ and has minimum total length.

The TSP has also been formulated as an ILP problem using different formulations. One of the first ILP formulations is ascribed to Dantzig et al. [126], and it is defined as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} && && (2.5)\\
\text{subject to} \quad & 0 \leq x_{ij} \leq 1 && \forall i, j \in \{1, \dots, n\} && (a)\\
& \sum_{i=1, i \neq j}^{n} x_{ij} = 1 && \forall j \in \{1, \dots, n\} && (b)\\
& \sum_{j=1, j \neq i}^{n} x_{ij} = 1 && \forall i \in \{1, \dots, n\} && (c)\\
& \sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |Q| - 1 && \forall Q \not\subset \{1, \dots, n\}, |Q| \geq 2 && (d)
\end{aligned}$$

Here, the constraint (a) restricts the decision variable x_{ij} to be binary ($x_{ij} = 1$ means that there is an edge(i, j), and 0 otherwise), the inequality constraint (b) expresses the fact that exactly one edge enters each node, the (c) that exactly one edge leaves each node and (d) ensures that there are not sub-tours among the non-starting vertices.

Traditionally, the TSP has been classified in symmetric Travelling Salesman Problem (sTSP) and asymmetric Travelling Salesman Problem (aTSP). That is, if we have a distance d_{ij} between the node i and the node j, in the sTSP $d_{ij} = d_{ji}$, and in the aTSP $d_{ij} \neq d_{ji}$. However, both problems have had a parallel evolution because of their similarities. Exact algorithms were the first approaches with which researchers started to study both problems. These approaches have always been incidental with the ILP formulation. In this way, the branch-and-bound paradigm appeared in 1976 to solve the sTSP with the work of Miliotis [127]. This paradigm consists of relaxing some problem constraints and then regaining feasibility through an enumeration process by means of a search space. Some of the best known aTSP algorithms are the developed by Eastman [128], Little et al. [129] or Carpaneto and Toth [130]. Other approach is the branch-and-cut, which involves running a branch-and-bound algorithm and a cutting plane method to tighten linear programming relaxations. This cutting plane methods [131] helped to design branch-and-cut algorithms [132, 133] to overcome large sTSP instances. It was in 2003 when Applegate et al. [134] published the Concorde solver, which is a computer program that combines some of these approaches to obtain feasible solutions even in extreme large instances.

Another line of research for the TSP has been the heuristics algorithms, which deliver approximate solutions in reasonable time. Laporte [135] states that the heuristic design has two main streams: heuristics focused on guaranteeing a worst-case performance [136, 137] and those focused on a good empirical performance. In this regard, here exist two main approaches related to design heuristics with a good empirical performance: tour construction and tour improvement approaches. A tour construction heuristic iteratively builds a solution by adding a new vertex at each step such as the nearest neighbour heuristic [138], insertion algorithms [138, 139] or the Christofide heuristic [137]. Otherwise, a tour improvement heuristic enhances a solution by performing an additional improvement step such as the r-opt algorithm [140], the Lin-Kernighan heuristic [141], simulated annealing algorithms [142, 143] or tabu search [144]. Additionally, the randomized improvement has also been studied in the last decades. These heuristics use randomized algorithms, such as genetic algorithms [145] or ant colony algorithms [90] to obtain nearly optimal solutions. The recent book of Davendra [146] provides an exhaustive and updated collection of the state of the art about the techniques and algorithms that solves the TSP and its generalizations.

A well-studied generalization is the mTSP, which consists of finding tours for *m* salesman, who all start and end at the depot, such that each intermediate node is visited exactly once and the total cost of visiting all nodes is minimized. The first algorithm to solve the mTSP was published by Laporte and Nobert [147]. Since then, several approaches have been proposed as exact algorithms [148, 149] or heuristic solutions [141, 150]. There also exists important variations of the mTSP, such as the multiple Travelling Salesman Problem with Time Windows (mTSPTW) [34], where the nodes need to be visited in a particular time interval. Also, the mTSP is considered as a relaxation of the Vehicle Routing Problem (VRP) but without the capacity constraint of the vehicles.

The work of Dantzig and Ramser [151] proposed the first formulation of the VRP, called as the truck dispatching problem, where the goal is to route a group of gasoline delivery trucks between a bulk terminal and a large number of service stations. Laporte [33] explains some of the most important exact and approximate algorithms developed for the VRP. As well as for the mTSP, there are generalizations of the VRP, such as when the length of any route may not exceed a prescribed bound (Distance-constrained Vehicle Routing Problem (DVRP)) [152], where a node *i* needs to be visited before node *j*. We encourage the reader to take a look at the work of Braekers et al. [153], which contains an updated and taxonomic overview of the literature published in the last ten years about the VRP.

We have already presented an overview of the optimization problems and algorithms existing in the literature that are applied in the robotics engineering field. Also, we described the particular problems and algorithms that have been under study in this PhD. So, in the next section we will talk about the evolution of cooperative multi-robot systems which formulate discrete optimization problems, and integrate optimization algorithms to solve them.

2.2 HETEROGENEOUS COOPERATIVE MULTI-ROBOT SYSTEMS

Multi-robot systems were introduced in the literature during the 1980s [154, 155], but the technological advances at that time delayed their growth in engineering applications till the late 90's. As Figure 2.2 shows, the scene on multi-robot systems in the robotic engineering field has shifted completely during the past three decades. The technological advances have been relevant for the rising of several mobile robotic systems such as: the Unmanned Aerial Vehicle (UAV), the Autonomous Underwater Vehicle (AUV) or the Unmanned Ground Vehicle (UGV). In particular, UAVs have experienced a research explosion from less than seven thousand works during the 90's till more than thirty thousand in the last decade. The expansion in single-robot systems (UAVs, UGVs or AUVs) might have triggered the investigation on multi-robot systems, as they have experienced a notable growth ever since. Figure 2.2 shows a clear lineal evolution in multi-robot systems research, similar to the linear evolution of the other single systems.

The fundamental idea behind the multi-robot systems is in the divide-and-conquer approach, dividing a high-level problem into sub-problems and then, dispatching these sub-problems to individual robots of a team and allowing interactions among them to achieve complex tasks. Thus, the cornerstone of multi-robot systems is not in

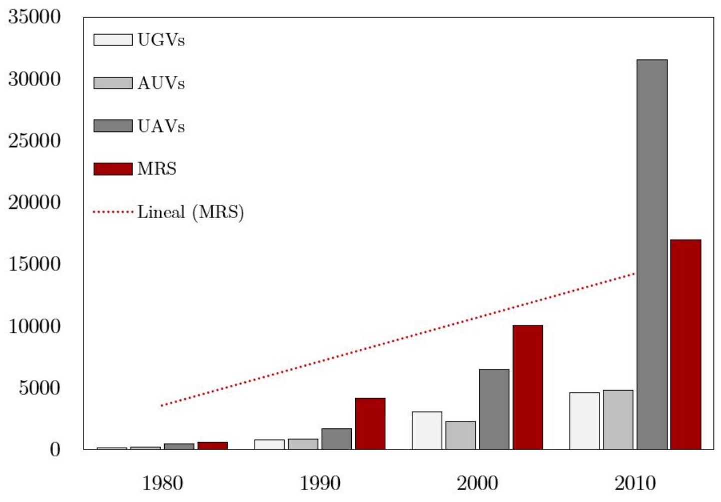


Figure 2.2: Research works progression in UAVs, UGVs, AUVs and Multi-Robot Systems (MRS) in the literature from 1980 to 2019. Data obtained from the Web of Science database. Each year shows the number of works (at 10-year intervals) on each topic following the boolean expressions: TS=(uav* OR drone* OR MAV*) for UAVs, TS=(ugv* OR rover*) for UGVs, TS=(auv*) for AUVs and TS=("Multi-Robot system" or MRS or "multiple robots") for MRS.

the individual complex capabilities but in the cooperation synergies exploited by them.

Historically, the cooperation in multi-robot systems has been considered a sub-category of a general term known as collective behaviour [156, 157]. From a sociology perspective, Giddings [158] defines a collective behaviour as *social processes and events which do not reflect existing social structure (such as laws), but which emerge in an spontaneous way*. From the robotic perspective, Yan et al. [157] summarize a collective behaviour as any behaviour of agents in a system having more than one agent. In this way, following other definitions of cooperation in the robotics literature, Cao et al. [156] define the cooperation behaviour in multi-robot systems as follows: *given some task specified by a designer, a multi-robot system displays cooperative behaviour if, due to some underlying cooperation mechanism, there is an increase in the total utility of the system*. Nevertheless, the apparent improvement of multi-robot systems puts additional burden in setting up or deploying such systems as they can crash with higher probability during cooperation than single systems.

On this line, we can state that cooperation implies integrating additional complex capabilities into the multi-robot system, such as a communication layer or a robust contingency logic, aiming to achieve complex tasks that for single-robot systems are not possible. For instance, single-robot systems cannot perform spatially separated tasks that require coordination in a brief period. Another example is a set of tasks which requires robots with different capabilities, such as performing oceanographic exploration with AUVs whereas a group of

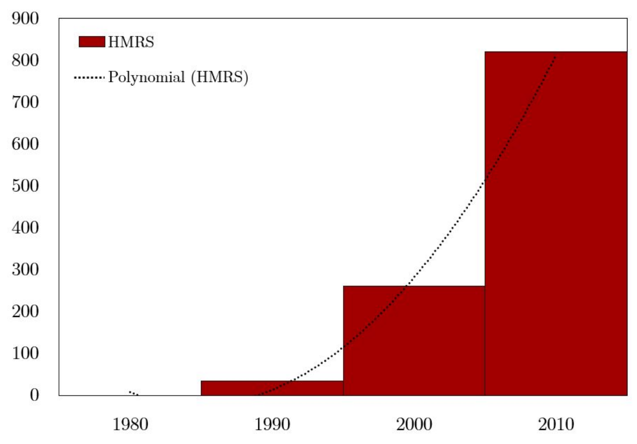


Figure 2.3: Research works progression in heterogeneous multi-robot systems (HMRS) from 1980 to 2019. Data obtained from the Web of Science database. Each year shows the number of works (at 10-year intervals) following the boolean expression: TS=(("Multi-Robot system" or MRS or "multiple robots" or UAV* or UGV* or AUV* or MAV*) and heterogeneous).

UAVs perform reconnaissance flights to study the atmospheric conditions at the same time, or drilling the terrain with a UGV while a group of UAVs perform exploration flights to determine future interesting places to drill. Here, we can devise a wide variety of multi-robot configurations that can be set up in different ways to deal with any cooperative problem.

Several works in the literature have dedicated efforts to build a complete taxonomy of these multi-robot configurations under different taxonomic axes [159–161]. Although their classifications are based on criteria approached from different foundations, most of them are in agreement with a particular taxonomic axe: the multi-robot system composition. The composition axe classifies the multi-robot systems based on the capabilities differentiation among the robots. There are two classes: homogeneous and heterogeneous. That is, if the physical capabilities of the individual robots are identical, the multi-robot system is homogeneous. Otherwise, if the capabilities of the robots are different, i. e., suitable to perform different tasks, the system is heterogeneous.

Nevertheless, we found that the current state of the art about heterogeneous systems has been increasing exponentially during the last decade (see Figure 2.3), and a more subtle classification of these systems might be appropriate. For this thesis, in the shake of clarity, we propose two new classes to classify heterogeneous multi-robot systems: simple and multiple. That is, if the heterogeneous system has one robot per category, where a category defines one robot with a particular set of capabilities, it is a simple system. For instance, if the robot UGV$_a$ and the robot UGV$_b$ are in different categories, i. e., form a heterogeneous simple system, we notate the system as: UGV$_a$-

Table 2.3: A brief classification of different multi-robot systems configurations as heterogeneous simple systems or heterogeneous multiple systems. Note that multiple UAVs and multiple UGVs are configurations where at least two of them are different.

Class	Sub-class	Configuration
Heterogeneous Multi-Robot Systems	Simple	UGV-UAV [162–165]
		AUV-UAV [166–168]
		ASV-UAV-AUV [169, 170]
	Multiple	UGVs [171–173]
		UAVs [174–176]
		UAV-AUVs [177, 178]
		UGV-UAVs [179, 180]
		UAV-UGVs [181, 182]
		UAVs-UGVs [183, 184]

UGV_b. Examples of simple systems are: UGV-UAV (UGV and UAV are clearly in different categories), UAV-AUV-UGV or UGV_1-UGV_2-UAV, where UGV_1 is not in the same category than UGV_2 because they have different capabilities. Otherwise, if the heterogeneous system has more than one robot in at least one category, it is a multiple system. Examples of multiple systems are: UAVs, UGV-UAVs, UGV_1-UGV_2s or UAVs-UGV_1-UGV_2; where all of them have at least one category with more than one robot. We have collected some existing works in the recent literature (see Table 2.3) following our proposed classification. Here, if the reader goes further, he can appreciate that the average of the years of publication of the collected works is 2014, which is another demonstration of the growth of multi-robot systems in the last decade.

In the following subsection 2.2.1 and subsection 2.2.2, we provide a detailed description about the state of the art of the two heterogeneous multi-robot configurations studied in this PhD: simple UGV-UAV and multiple UGV-UAVs systems.

2.2.1 *Simple UGV-UAV systems*

The heterogeneous simple UGV-UAV system was one of the first multi-robot systems to be explored, because of their simplicity (considering the inherent complexity of multi-robot systems) and the technological advances in UAVs. The industry quickly became aware of the rise of these systems and started proposing cooperative solutions to old complex problems. Here, we can highlight the following three widely studied problems deploying heterogeneous simple UGV-UAV systems: guidance, persistent surveillance and exploration problems.

The guidance problem is based on deploying a robotic system for assisting in the navigation process to others systems. Usually, the UAV is the robotic system chosen to collect aerial information to guide the UGV on the ground. Sofman et al. [185] describe methods to extract relevant information from the environment using a UAV to enhance the navigation of the UGV. Cantelli et al. [186] propose a cooperative strategy where the UAV runs a vision tracking algorithm to automatically follow the UGV, whereas is providing environmental images of the terrain to the UGV, aiming to build 3D maps and gather photogrammetry data. Mueggler et al. [187] present a path planner in which the UAV scans the area and searches for the fastest route for the UGV in order to deliver a first-aid kit. Harik et al. [188] propose a guidance scheme in which a UAV provides target points to the leader UGV which carries objects in unsafe industrial areas. These methods share a particular characteristic, that is, all of them do not have information about the environment off-line, so the UAV has to previously collect aerial data of the terrain to provide on-line support for the routing of the UGV.

The surveillance and patrolling problem is another well-studied problem whose goal is to guarantee a proper safety level to a specific target in a controlled area. Saska et al. [189] introduce a UGV-UAV system to perform periodical surveillance in indoor environments. They propose the UGV as a moving station that carries the UAV along the terrain and, so, the UAV as the flying inspector. Reardon and Fink [190] describe a cooperative UGV-UAV system to identify threats to human safety by patrolling the team's surroundings, identifying threats, and notifying the human team member. They formulate the problem as a three-dimensional surveillance task, which generalizes the art gallery problem and the TSP explained in section 2.1. Manyam et al. [163] define the cooperative air-ground vehicle routing problem in which a heterogeneous simple UGV-UAV system is involved. The objective is to visit a set of targets either by the UAV or the UGV, keeping alive the communication link between both vehicles. As the reader can appreciate, the cooperative approach in these works defines the UAV as the aerial vigilant and the UGV as the ground marshal.

The exploration problem focuses on reaching a set of targets on a undetermined area, and it is referred to find a path for the multi-robot system optimizing the overall mission time. The problem of finding an optimal solution among a set of target points is the TSP (see subsection 2.1.3). Papachristos and Tzes [191] present a cooperation where the UGV-UAV team is connected through a power-tethering physical link, aiming to address the challenge of large-scale exploration missions. In this way, the team explores the unknown environment, while incrementally is building the explored world by using sampling-based trajectory planning techniques. Maini and Sujit [192] formulate the Fuel Constrained UAV Refuelling Problem with Mobile Refuelling Station, in which the UAV has to reach the target points and it uses the UGV as a moving charging station to place charging stops around the UGV's path. This formulation derives from a cooperation paradigm in which the strategy is: firstly, formulating the SCP (see subsection 2.1.2) to select the optimal number of UGV's locations, and secondly, modelling the path planning problem as TSPs. Hood et al. [193] propose a cooperative exploration where the UAV holds the same relative position than the UGV but with a certain altitude, providing to the UGV a birds-eye view beyond terrain obstacles that hinders the exploration to the UGV. Yu et al. [194] study the problem of routing an energy-limited UAV to visit a set of places in the least amount of time. They consider path planning scenarios where the UAV can recharge by landing on stationary recharging stations or on the UGV acting as a mobile charging station.

Nowadays, the list of problems under study deploying these systems keeps growing due to the great cooperation synergies of simple UGV-UAV systems. For instance, we can mention their research in surveying operations [195], precision agriculture [162], wildfire detection [196] or last-mile delivery [4, 197]. Therefore, we can ensure a bright future for simple UGV-UAV systems.

2.2.2 *Multiple UGV-UAVs systems*

The multiple UGV-UAVs systems have been experiencing a similar progress during the last-decade. A multiple UGV-UAVs system is formed by a single UGV and multiple UAVs. These systems have been studied in analogous problems, but integrating more UAVs aiming to improve the efficiency of the solution on different metrics. For instance, maximizing the area covered by partitioning the space into regions to be covered by different UAVs, reducing the exploration time by splitting the target points among the UAVs or minimizing the delivery time of parcels by parallel dispatching among the UAVs. These examples represent the following three problems on the investigation of heterogeneous multiple UGV-UAVs systems: surveillance and patrolling, exploration and last-mile delivery.

The surveillance and patrolling problem has also been studied by deploying multiple UGV-UAVs systems. We can find in the literature that this problem is also known as Intelligence, Surveillance and Reconnaissance (ISR) mission. Primarily, the idea of the ISR mission is the coordinated acquisition, processing and provision of coherent and relevant information to support human operator's activities. Pippin et al. [198] build a decentralized task assignment and collaboration platform between multiple UAVs and a car sized UGV in target detection and surveillance tasks. They perform field experiments where a team of UAVs explore the area looking for a target location. When one of the UAVs locates the target, it sends a message to the team members with the target location, so they can navigate to that location and perform surveillance. Hager et al. [199] present a cooperative system formed by two UAVs, one UGV and a set of stationary ground sensors. The UGV is in charge of verifying the identity of a target thanks to its equipped infra-red sensor. Then, it cooperates with the UAVs surveillance assets by sharing the predicted target location, and helping to improve the accuracy of the target localization estimate through additional measurements. Çaşka and Gayretli [200] study a cooperative system formed by a UGV carrying a couple of UAVs, aiming to perform persistent ISR missions. They developed an algorithm which dispatches tasks to the UAVs considering their battery levels.

As well as the surveillance problem, the exploration problem has also been studied in both single UGV-UAV and multiple UGV-UAVs configurations. Kim et al. [201] propose a couple of UAVs as a stereo vision system to assist global path planning for a UGV in GPS-denied environments. The UAVs follow the UGV by recognising a marker on the top of the UGV, and also, they are in charge of building a depth map of ground objects aiming to provide detailed information of ground obstacles to the UGV. Mas et al. [202] address a leader-follower approach, based on dual quaternion representations, where the UAVs keep a formation to escort the UGV through the environment. Shi et al. [203] present a similar leader-follower approach where the leader is the UGV and the UAVs are the followers. The UAVs have to keep a desired formation to track the trajectory of the leader. Their approach is demonstrated by a system of three UAVs and one UGV. Chen et al. [204] describe a cooperative path planning problem consisting of one UGV and two UAVs. The UGV carries the UAVs through the environment and the UAVs can take off and land on the carrier for its energy-saving and recharging tasks. The path planning problem for the multiple UGV-UAVs system is modelled as a multi-constrained optimization problem, then they integrate a particle swarm optimization algorithm to compute the solution.

As we show in the last paragraph of the above section, the last-mile delivery problem has been recently studied using single UGV-UAV systems. In fact, the literature about this topic has experienced an explosion since logistic companies such as Amazon [205], DHL [206], UPS [207] or FedEx [208], declared in 2013 [209] their intentions on developing new delivery paradigms using simple/multiple UGV-UAV systems to accomplish the arisen last-mile delivery problem. Their proposal describes the UGV as a truck which carries the parcels and a set of UAVs through the streets of a city. So, the UAVs deliver the parcels by taking off and landing on the truck. Among all the works that have emerged since then, we highlight the work of Murray and Chu [4], which is considered one of the first dealing with the real last-mile delivery problem. They formulated the Flying Sidekick Salesman Problem (FSTSP), where a set of customers, each of whom must be served exactly once by either a driver-operated delivery truck or a UAV operating in coordination with the truck. The reader can appreciate that the UGV is not fully autonomous in this problem and the heterogeneous system proposed is based on a simple UGV-UAV. However, over the last four years several works have emerged addressing this topic from the multiple UGV-UAVs systems perspective. This is the reason why we tackle the problem as a multiple system not as a simple system.

The work of Ferrandez et al. [5] studies the multiple UGV-UAVs for the last-mile delivery problem. They design a clustering k-means algorithm to find suitable locations, from where to perform the deliveries, and required drones per truck. Also, they implement a genetic algorithm to compute the truck route as a TSP. They also compare their method with a stand-alone delivery method, demonstrating that multiple drones are more optimal in time and energy terms. Motivated by this, Tu et al. [6] define an extension of the Travelling Salesman Problem with Drone (TSP-D) [210] but with a set of UAVs, called as Travelling Salesman Problem with multiple Drones (TSP-mD). Furthermore, they propose two algorithms to solve the TSP-mD; the first implements a greedy randomized adaptive search procedure and the second, an heuristic using adaptive large neighbourhood search. Murray and Raj [179] formulate the multiple Flying Sidekick Salesman Problem (mFSTSP) arisen from their previous work [4]. They formulate the problem as a Mixed Integer Linear Programming (MILP), and present a three-phased heuristic approach that provides solutions in reasonable time. Their MILP formulation models a queue scheduling for UAV arrivals and departures synchronization during the deliveries, and the flight endurance of each UAV as a function of the UAV's battery size, payload, travel distance and flight phases. Also, they analyse advantages and disadvantages of using the UAVs instead of the truck to perform the deliveries, and the benefits of adding more UAVs to an existing fleet. Peng et al. [211] propose a hybrid genetic algorithm that allows multiple UAVs carried by a truck to simultaneously deliver multiple parcels in different locations. The algorithm is split in three parts: population initialization, crossover and education. They demonstrate that their algorithm outperforms some existing heuristic algorithms.

Despite of all of the existing works in the last-mile delivery problem, most of them consider the UGV as a human-driven delivery truck, so much efforts are required to provide a high-level of autonomy to these approaches and to consider a real heterogeneous multiple UGV-UAVs system.

Nevertheless, the existing literature about heterogeneous cooperative multi-robot systems is extensive because of their impact and direct applications into real problems. In the next section, we will talk about autonomous controllers which model these cooperative multi-robot systems into high-level control architectures to operate a working team of mobile robotic systems.

2.3 AUTONOMOUS CONTROLLERS FOR MULTI-ROBOT COOPERATION

Autonomous controllers represent a current research topic devising high-level frameworks for enabling the cooperation in multi-robot systems. Numerous works on autonomous controllers [212, 213] present studies toward a technology to tackle challenges with a cooperative robots team. Nevertheless, before to start describing cooperative autonomous controllers, we want to introduce how these controllers evolved from a simple-robot perspective. Thus, in the next section we present when autonomous controllers became important in the research community and what are the main disciplines over which autonomous controllers have been evolved. Then, we will give special attention to the main ingredients (deliberation and reactivity) of an autonomous controller which results in different kind of these architectures. Finally, we will focus on autonomous controllers designed for multi-robot cooperation, and we will describe a well-known taxonomy to explain the main features to consider in order to design an autonomous controller for multi-robot cooperation.

2.3.1 *A brief introduction to autonomous controllers*

The field of distributed robotics has its origins in the late 80s, when several researchers began investigating issues in cooperative controllers for robotics systems [214–216]. Prior to this time, coordination and interactions of multiple intelligent agents was concentrated on either single systems or distributed problem-solving systems that did not involve robotic components. For instance, the book of Bond and Gasser [217] described several problems and techniques in the distributed Artificial Intelligence (AI) field. However, this book referred to distributed cooperation of software agents not mobile robots. As we said, it was in the late 80s when researchers started to work on cooperative mobile robots and the cooperative behaviour in robots took on a meaning of its own. The works of Fukuda and Nakagawa [214], Beni [215] and Asama et al. [216] represented initial approaches in autonomous controllers for multi-robot cooperation. Since then, many researchers shifted their focus from single-robot systems to cooperative multi-robot systems. Nevertheless, the technological constraints at the time was a major obstacle in the development of physical cooperative multi-robot solutions.

However, the technological advances in robotic systems did not came till the late 90's (see Figure 2.2), when robotic systems such as UGVs, UAVs or AUVs were becoming popular in the research community. It was then that an exponential growth of applied robotic solutions in various fields of application began. Autonomous controllers were a consequence of the rising of these robotics solutions. Cao et al. [156] presented in 1995 a genuine review about the evolution of multi-robot systems at the time, and they envisioned some promising directions for the cooperative robotics science: (1) Distributed AI, concerning to the study of distributed systems of intelligent agents. This discipline is split in distributed problem solving (concerned in solving a single problem using many agents) and multi-agent systems (studies the collective behaviour of a group of agents with possible goal conflicts); (2) Distributed systems, which is mostly related to the study of multi-robot systems from the distributed computation perspective, and so, the study of communication issues arisen from this distribution; and (3) Biological systems, which are referred to the study of collective behaviours in biological beings such as ants or bees. After more than 20 years, we can state that the cooperative autonomous controllers, devised so far, are encapsulated in these disciplines. Our research during this PhD has been focused on distributed systems, more specifically, in the study of autonomous controllers devised for heterogeneous cooperative multi-robot systems.

2.3.2 *Deliberation vs Reactivity in intelligent agents*

The concept of a controller in control theory is identical to that of an agent in AI [218]

Autonomous controllers can be defined as software architectures with a well-defined structure to control intelligent agents. Russell and Norvig [218] define an intelligent agent as *anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators*. They described four basic agent's structures: simple reflex agents, model-based reflex agents, goal-based agents and utility-based agents. In the following we will briefly describe them, in order to highlight on their deliberative and reactive capabilities.

Simple reflex agents are purely reactive agents that perceive the environment and directly select the action to be performed without considering the perceived history. So, the correct decision of these agents are only made if the environment is fully observable, which not always can be achieved. Thus, model-based reflex agents arise to handle partial observable environments. These agents use a model of the world to update the world-knowledge of the environment and how the robot can interact with the updated world. However, they choose an action in the same reactive way as the reflex agents, which not always guarantee that the chosen action is the correct decision. These decisions are usually subordinated to the achievement of a future state or goal. Then, the goal-based agents born to integrate the goal information with the states that the agent has to achieve. This goal information allows the agent to execute a decision-making process to deliberate the most appropriate action. If the decision-making process integrates a particular function to select the action that improves the performance of the agent, we have a utility-based agent. Here lies the main difference between reactivity (reflex agents) and deliberation (goal-based or utility agents). A reactive agent just senses the environment and executes the action which satisfies a simple condition, i. e., their decision-making process follows a sense-act cycle. Otherwise, a deliberative agent senses the environment, deliberates a plan and executes the plan to achieve a goal, i. e., their decision-making process follows a sense-plan-act cycle.

The reader must note that deliberative agents are not always more desirable than reactive agents. For instance, lets have a UAV in an initial location, with a constant speed and partial knowledge of the environment. The goal of the UAV is to achieve a desired location. At first, the UAV plans a route considering the partial knowledge of the environment (such as buildings) and starts flying. Then, the UAV senses an obstacle on its path (such as a high-voltage tower), so it has to launch a decision-making process to choose the correct decision in order to avoid that obstacle. If the decision-making process is too complex (such as deliberating a new complete route to the goal), it can require too much computational time to select the appropriate action before colliding with the obstacle. In this case, a reactive behaviour is desirable in order to make a decision in a minimal response time. In this regard, deliberative agents are designed to make decisions that do not require short response times. However, we want to point out that it is often a design decision of the developers to select the most appropriate agent, considering the possibility of coupling both abilities in a hybrid agent.

2.3.3 *Autonomous controllers for mobile robotic agents*

According to the above definition of intelligent agents, we can describe a robot as a physical intelligent agent equipped with sensors to perceive their environment and actuators to execute actions on the environment [218]. In particular, mobile robots move over their environment using mechanisms such as wheels, propellers or legs. Thus, in order to perform a desired task, it is essential for an intelligent mobile robot to have the abilities to sense and perceive the working environment, make a decision (reactive or deliberative) and execute an appropriate action.

An autonomous controller (also known as autonomous control architectures) define which abilities should be integrated into an agent (in this case, a mobile robot) to get desired results with different levels of autonomy[5]. That is, these architectures must decide how to combine reactive control and deliberative planning to get a feasible solution for a particular situation. The existing literature shows a wide spectre of autonomous controllers combining these abilities in several manners. In fact, a lot of literature have been dedicated to classify these control architectures by their system composition as follows: behaviour-based (reactive), deliberative, hybrid (reactive + deliberative) or multi-agent architectures.

Behaviour-based or reactive architectures are controllers which behaviour is determined by a reflex agent. Thus, these controllers follow a sense-act cycle, that is, do not maintain a model of the world and their decision-making process is based on a collection of predefined actions in order to fulfil the sensed state. One of the most cited behaviour-based controllers is the Subsumption architecture [220]. This architecture follows a reactive approach with different levels of competence, being the base level the most reflexive and the higher levels the ones including more complex behaviours. Thus, it represents a scalable model, because the lower levels represent elemental behaviours and the upper levels are complex behaviours attached to them, subsuming the roles of the lower levels. The Reactive Action Package (RAP) [221] controller is another well-known reactive controller, and it is based on executing, monitoring and replanning actions from a reactive manner. This architecture proposes a structure made up of Reactive Action Packages (RAPs). Each RAP is like an independent unit pursuing its own goals.

These systems present a high performance for making low-level decisions in real time. Rejecting complex models of the world and deliberation capabilities, allow them to select the next action to perform in a fast sense-act cycle. However, the lack of world knowledge along with deliberative planning makes these systems vulnerable to high-level decisions that require a broad knowledge of the environment.

Deliberative architectures are the other side of the coin of reactive architectures. These controllers are based on goal-based agents and follow the sense-plan-act cycle. First, the robot senses its surrounding and creates a world model of the static environment by combining sensory information. Then, it employs planning to search the appropriate action toward the goal and generate a plan to execute. The first autonomous controllers comprising deliberative capabilities in mobile robotics were related to control the motion planning of the robots [222, 223].

Takahashi and Schilling [93] define a motion planning controller for a robot geometrically modelled as a rectangle, in a environment with polygonal obstacles. They implement a technique that use a

5 The European Cooperation for Space Standardization (ECSS) [219] defines four autonomy levels E1-E4. Autonomous controllers are directly related to the highest level E4.

generalized Voronoi diagram to select the motion path among the environment. Nagatani et al. [224] also build in execution time a Voronoi graph of the environment thanks to the sensory information during the exploration. Yang et al. [225] propose a sophisticated fuzzy logic motion planning controller. This controller has two deliberation levels to control a mobile robot in unknown environments. At the highest level, a fuzzy logic planner merges the input goal with the sensor information to produce intermediate points that define the trajectory of the robot. At the lowest level, these points are taken as sub-goals, and along with a short-range sensory information, the fuzzy logic planner guides the robot to reach the sub-goal while avoiding collisions. Huq et al. [226] present a controller that models different robots schemas and use fuzzy logic to select the suited one based on the sensory information. High-level schemas use the world-knowledge to perform motion planning trajectories by partitioning the space using Voronoi diagrams. Otherwise, low-level schemas use current sensory data to detect obstacles and avoid them.

From deliberative controllers in mobile robotics, we can highlight the performance of Voronoi diagrams to build models of the world in execution time (section 2.1 describes some geometric problems for motion planning). However, deliberative navigation often requires an accurate model of the environment for planning global paths. For simple and static environments this might be feasible, but in complex and dynamic environments it is required computational resources and memory to deliberate a proper motion plan. Also, static and fuzzy control modules might not have all the predefined actions to response to a dynamic environment. Thus, these approaches may not have a proper performance in the presence of uncertainty in a real world.

In order to perform autonomous navigation in a real world, it is required reactive capabilities to provide reactivity in low-level decisions to face the dynamic world, and deliberative capabilities to integrate high-level decisions considering a complex model of the world. The controllers that merge both capabilities are called as hybrid architectures. The most popular schema of hybrid controllers is the three-layer architecture, which consists of a deliberative layer at the top-level, an executive layer at the middle level and a reactive layer at the bottom level. The deliberative layer provides a high-level decision making process to deliberate over complex global goals. It is usually formed by a planner, which uses predefined models of the world and the robot system to compute a global plan. The executive layer guarantees a proper execution of the generated plan following a predefined execution approach. Then, the reactive layer contains reactive behaviours and sensing capabilities to provide a fast sense-act cycle to the robot.

A high-level deliberation process usually takes minutes to compute a plan

A low-level control usually takes milliseconds to select the appropriate action

The Bonasso 3-Tiers architecture [227] is a well-known hybrid architecture which follows a three-layer scheme with: the skill layer, the sequencing layer, and the planning layer. The skill layer is based on reactive control loops where the sensors and actuators are tightly coupled. The sequencing layer provides an execution cycle that selects the next behaviour to execute by the skill layer. The planning layer is defined to execute in real-time deliberative algorithms (such as search algorithms) to compute global plans for the system. As Gat et al. [227] states, the key idea of this approach is to perform deliberative planning while the lower layers follow a nominal execution. Other examples of hybrid layered architectures are: A Three-Layered Architecture for Navigating Through Intricate Situations (ATLANTIS) [228], Laboratory of Analysis and Architecture of Systems (LAAS) [229] or Couple Layered Architecture for Robotic Autonomy (CLARAty) [230].

An evolution of the layered architectures are the multi-agent architectures, which are hybrid approaches where the architecture is decomposed into agents that can integrate reactive, deliberative or both. The decomposition into agents follows a divide-and-conquer approach, where each agent deals with a particular problem and the combination with other agents allows the system to achieve complex tasks.

The Intelligent Distributed Execution Architecture (IDEA) [231] developed at NASA in 2002 is an example of multi-agent architectures. IDEA interleaves deliberative and execution capabilities into the same framework by sharing a common database of knowledge. In this regard, a controller based in IDEA is a collection of agents, where each one has a particular purpose and the deliberation is distributed among them. Then, merging the solutions provided by the different agents, the full system is able to achieve a global goal.

The Teleo-Reactive EXecutive (TREX) [232] is an autonomous controller which follows the philosophy of the goal-based agents described in subsection 2.3.2. T-REX also interleaves deliberation and execution into a single agent. A T-REX agent is the coordinator of a set of control modules. Thus, each control module (also called reactor) encapsulates the sense-plan-act cycle in a closed-control loop to act over particular sub-goals of the global mission goal. The T-REX controller is the multi-agent architecture that has been addressed in this PhD for the deployment of multi-robot cooperation.

Although any kind of the presented system compositions can be applied to build autonomous controllers for multi-robot cooperation, multi-agent architectures might represent the most suitable approaches for the following reasons: first, they combine the deliberation and reactive capabilities as well as the hybrid architectures, which gives them all the advantages of the first three systems compositions; and second, they introduce a system decomposition into agents and a communication framework that promotes them for their deployment in distributed systems. That is, each agent is able to represent a hybrid system into itself, meaning that can be used to control a single robot system[6]. As well as intelligent agents, the design of cooperative autonomous controllers is a developer's decision. Thus, in the next section, we will see how cooperation affects the design and classification of autonomous controllers for multi-robot systems.

2.3.4 *Cooperation in autonomous controllers*

Autonomous controllers for multi-robot cooperation are mostly built on top of the system compositions presented in the previous subsection. However, these cooperative controllers cannot be simple regarded as a generalization of single-robot controllers, because the embedding of coordination capabilities implies addressing additional questions. These questions are discussed in the literature [213, 233] as the relevant features that may characterize a cooperative controller. Iocchi et al. [233] propose a taxonomy that classifies cooperative autonomous controllers from these different features. This taxonomy splits up cooperative autonomous controllers into four levels (see Figure 2.4): the cooperation level, the knowledge level, the coordination level and the organization level. In the following we will describe each of these levels and some well-known autonomous controllers in the literature that deploy cooperation mechanisms.

At the cooperation level, Iocchi et al. [233] follow the cooperation definition in Noreils [234] to distinguish between cooperative and non cooperative systems. They describe cooperation as the scenario where a team of robots operate together by performing individual tasks to achieve a common goal and improving the performance of a single-robot system[7]. Once an autonomous controller has been categorized as cooperative, the knowledge level focuses on the robots team by defining the awareness feature [235]. Awareness is the property of a robot to have some kind of knowledge of the other members of the team, i. e., the perception of other robots locations and actions. There can be aware and unaware systems.

At the coordination level, there are different coordination modes among the aware controllers. In this way, the literature [233, 236] generally defines coordination as a form of cooperation where the planned actions for some robots take into account the executed actions of others robots of the team, so there is a coherent synchronization among the actions that allow to achieve complex tasks. It is important to note that this can be cooperation without coordination. There are two kinds of coordination (besides the non coordinated systems): strongly or weakly coordinated. Strongly coordinated refers to a specific coordination protocol that governs a stiff behaviour of the robots team.

6 Obviously, this is not a rule, because an agent can be created to control the cooperation of multiple robots in a centralized manner.

7 Note the similarities with the cooperation definition described in section 2.2.

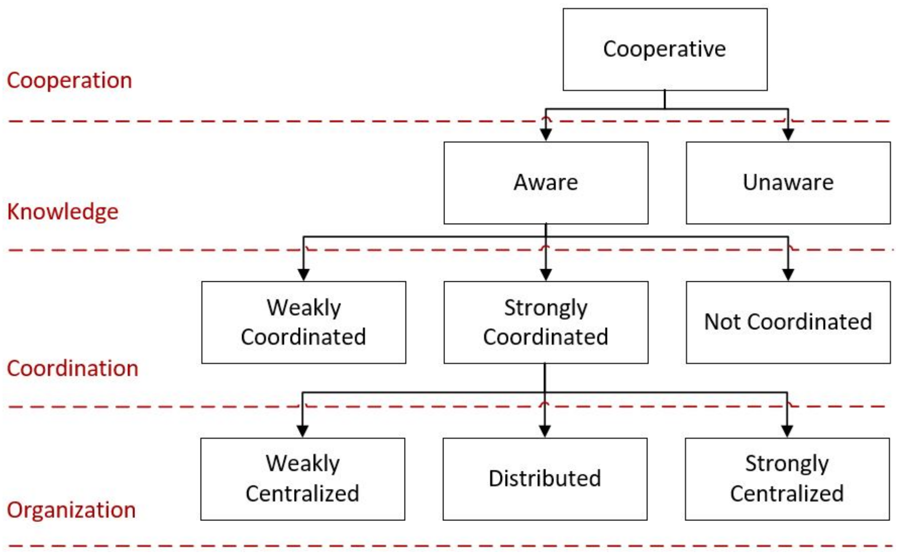


Figure 2.4: Multi-robot systems taxonomy proposed by Iocchi et al. [233], focusing on their cooperative features.

Otherwise, weak coordination implements a relaxed coordination without a particular protocol.

The lowest level of this taxonomy is the organization level, which refers to the classification of autonomous controllers based on the way the decision making process is organized within the controller [156]. At this level, we can distinguish between centralized and distributed approaches.

On one hand, centralized architectures name a robot as the central agent of the team. The central agent has the global world-knowledge of the environment as well as how every robot can interact with the world, and it is in charge of the decision making process of the whole team. Also, Iocchi et al. [233] split centralization in two kinds: strongly or weakly centralized. A controller is strongly centralized when the central agent remains the same during the mission duration, and weakly centralized when more than one robot in the team can become the central agent at any time during the mission duration.

Caloud et al. [237] present the GOFER controller, a typical centralized architecture, in which a group of mobile robots are managed from a central task planner, a motion planner and a scheduler. These planners have the world-knowledge of the environment and every robot in the team. Tang and Parker [238] describe the Automated Synthesis of Multi-robot Task solutions through software Reconfiguration (ASyMTRe), an architecture to synthesize valid and efficient multi-robot behaviours through the mapping of control schemas to manage a network of robots and accomplish a global task. The decision making system of ASyMTRe is centralized, and it is used to reconfigure the network. They demonstrate the performance of their approach in multi-robot transportation and box pushing problems.

Also, Milutinovi and Lima [239] describe a central controller which computes partial differential equations that describes the evolution of a team of robots under uncertain and dynamic environments. This central controller sends commands for task execution, task cancellation or task switching to every robot. Mas and Kitts [240] propose a framework for multi-robot formation control methods. This framework can be modelled as a centralized as well as a decentralized approach, and it is based on clustering techniques to control the motion of mobile multi-robot systems.

On the other hand, distributed architectures do not have a central node and the decision making process is distributed among all the team members. An example of this approach is the ALLIANCE architecture [241], a behaviour-based distributed architecture where a multi-robot team cooperates to perform missions based on independent tasks. Each robot can sense with some probability the effect of their own actions and the actions of other team members through perception and explicit broadcast communications. Then, each robot selects its own tasks, considering that those tasks are aligned for the benefit of the whole team. ALLIANCE uses motivational behaviours to allow robot team members to perform tasks only as long as the task demonstrated a significant advance toward the team's goal.

An architecture that follows a distributed decision making process is the High-level Distributed DecisioN (HiDDeN) [242], which is a distributed deliberative architecture that manages the execution of a hierarchical plan for a multiple robot team in a specific air–sea scenario. The hierarchical plan is distributed for its execution among the supervisor of the robots. Each robot has its own supervisor, which contains the robots actions and synchronization tasks with the others. Besides, the supervisors include a hierarchical repair process to provide fault tolerance to the system. Both hierarchical plan and repair processes are instantiated as hierarchical task networks [243], a common planning approach in cooperative architectures.

Despite the taxonomy depicted above, many architectures in the literature do not conform to a strict paradigm dichotomy, i. e., many architectures combine a distributed low-level decision process along with a centralized high-level deliberative system. These are called as hybrid architectures, and their fundamental objective is to combine local control with higher-level control to achieve both robustness and the ability to influence the entire team's actions through global shared goals. An example of hybrid architecture is the hierarchical architecture. The hierarchical architectures use to follow the leader-follower paradigm, where there can be various central nodes which control the decision making process of different sub-teams of robots, which in turn can control yet another groups of robots, and so forth, down to the lowest robot.

The work of Parker et al. [244] describes a cooperative architecture for the navigation assistance task. Here, the leaders are sensor-rich robots modelled to assist in the navigation of sensor-limited robots (followers) which do not have on-board capabilities for obstacle avoidance or localization. Other example is the CoT-ReX [245] controller, which is a hybrid architecture adapted to the problem of underwater detection and localization. It uses AUVs to gather information about the targets locations while the Autonomous Surface Vehicle (ASV) acts as a communication hub among all the AUVs. CoT-ReX has been built under the T-REX system for the planning and execution control of the mission.

2.4 SUMMARY

We started talking about the existing literature in discrete optimization problems and algorithms applied in robotics. In this first section, we described the nature of the discrete optimization and two of the most studies categories of optimization problems: combinatorial and proximity & visibility problems. We also talked about the NP-Completeness theory introduced in 1971 by Stephen Cook, and the evolved techniques since then, and we continued by describing polynomial time algorithms computing sub-optimal solutions to NP-Complete problems. In particular, we provided a detailed description of the combinatorial and proximity & visibility problems studied in this PhD: the NNS, the SCP and the TSP. We also mentioned some of their most common generalizations as well as existing algorithms to compute approximate solutions.

In a second section, we began talking about the evolution and the reasons of the expansion of multi-robot systems. We described the meaning embraced by the community for robotic cooperation, which erased from the collective behaviour concept. Then, we emphasized about the explosion of heterogeneous cooperative multi-robot systems in the last decade, and proposed a novel taxonomy for these systems. Furthermore, we provided a detailed description of simple and multiple UGV-UAV systems, because these have been the configurations studied in this PhD.

Finally, in the last section, we talked about the high-level frameworks designed to manage mobile robotic systems, known as autonomous controllers. We began with a brief historical introduction to these frameworks, then talked about the main elements that form them, i. e., deliberation and reactivity. We continued talking about the taxonomy of these systems from a single-agent perspective. And finally, we gave a brief overview of cooperative autonomous controllers, and described a well-known taxonomy in the literature proposed to classify different kinds of cooperation frameworks.

Part II

THE RESEARCH STUDIES

The exploration and observation of the environment represent a fundamental part of the scientific method. Then, an experimental stage elaborates the appropriate experiments and evaluates the obtained outcomes. In computer science, these experiments usually are embodied by algorithms, sequences of operations achieving computational solutions for the observed problems.

3 A COOPERATIVE SIMPLE UGV-UAV PATH PLANNING ALGORITHM FOR THE EXPLORATION PROBLEM

As we have discussed, heterogeneous simple UGV-UAV systems represent a vibrant research topic for cooperative exploration. Hence, encouraged by the future of cooperative UGV-UAV applications, such as the Mars 2020 Mission[1], we created a cooperative UGV-UAV path planning algorithm which exploits a UGV-UAV cooperation paradigm to tackle a particular exploration problem.

The existing literature shows a broad diversity of cooperative UGV-UAV exploration systems, but neither are conceived for high-level autonomous explorations (were both UGV and UAV are fully autonomous systems) nor are devised for particular exploration problems. In the next section we present a novel exploration problem which can be the basis of complex exploration problems. Then, we describe the cooperative path planning algorithm to solve the exploration problem. This algorithm implements a UGV-UAV cooperation paradigm by executing an ordered set of computational stages. In section 3.2 we provide an exhaustive description of the $\mathbb{R}^2$ path planning algorithm by explaining its different stages. Then, we present the extension to $\mathbb{R}^3$ exploration problems, and also, the modifications to build the equivalent $\mathbb{R}^3$ path planning algorithm. Finally, we provide an extensive experimental evaluation to characterize the path planning algorithm and the implications of the extension to three-dimensions.

3.1 THE EXPLORATION PROBLEM

The exploration problem that we propose it is formally named as the Energy Constrained UAV and Charging Station UGV Routing Problem (ECU-CSURP), and it is defined with the following constraints:

(i) An exploration area modelled as an $\mathbb{R}^2$ Euclidean space.

(ii) A set of target points which has to be visited.

(iii) A distance constraint where at least one target point is out of the boundaries of the initial UAV energy constraint.

(iv) An heterogeneous simple UGV-UAV system in which both have been modelled as Dubins vehicles [247], which are vehicles with just a single constraint: forward movement at a constant speed.

1 We recommend the reader to watch the keynote given by Dr. Mimi Aung in Caltech in 2015 [246].

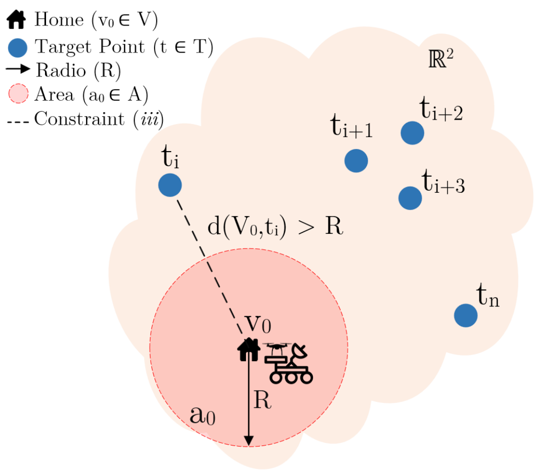


Figure 3.1: An ECU-CSURP instance where target points $(t_i, t_{i+1}, t_{i+2}, ..., t_n \in T)$ and a home location $(V_0 \in V)$ are distributed around a $\mathbb{R}^2$ Euclidean space. R denotes the farthest distance the UAV can travel considering a return flight to the take-off point, and $a_0 \in A$ is the area initially covered by V_0. Note the distance constraint is satisfied with $d(V_0, t_i) > R$.

(v) A UAV energy constraint.

(vi) A home location where both systems start and end the exploration.

The objective of the ECU-CSURP is to find a cooperative routing for the simple UGV-UAV system to allow the UAV to visit every target point while trying to minimize the overall travelling distance. Figure 3.1 shows an ECU-CSURP instance with a graphical representation of the problem constraints[2]. We model the $\mathbb{R}^2$ Euclidean space as an area where the distance travelled by both robotic systems is directly proportional to the time spent and the energy consumed in a trip. Therefore, the shorter the distance travelled, the shorter the time spent and the lower the energy consumed, and vice versa. Furthermore, the UAV energy constraint is modelled as the maximum distance the UAV can travel with a fully charged battery. Let V_{uav} be the UAV's constant velocity and t_{trip} the UAV's flight time computed in a trip with a fully charged battery, where $d_{max} = (V_{uav} * t_{trip})$. Thus, the farthest distance the UAV can travel ensuring a return flight to the take off point is $R = d_{max}/2$ (see Figure 3.1). Also, we assume that the UGV does not have energy constraints, so it has enough energy resources to complete the exploration mission.

Let T denote the set of targets points $\{t_1, ..., t_n\}$ where a target $t_i \in T$ for $i = 1, ..., n$, and V denote the set of charging stops, also called as vertices, $\{v_0, ..., v_m\}$ where a vertex $v_j \in V$ for $j = 0, ..., m$

2 We will rigorously follow this ECU-CSURP instance to explain every stage of the algorithm.

(see Figure 3.2). Let $v_0 \in V$ be the home location and let a UGV's path be denoted as a tuple $\{v_0, ..., v_{m-1}, v_m, v_0\}$. Then, let the distance constraint be represented as $\{t \in T : d(V_0, t) > R\} \neq \emptyset$. Let A denote the set of areas $\{a_0, ..., a_m\}$ where an area $a_j \in A$ (taking R as the radius) represents the set of covered target points $\{t_1, ..., t_n\}$ by $v_j \in V$. Let f_{v_i,v_j} represent the travel cost as the distance travelled by the UGV to travel from $v_i \in V$ to $v_j \in V$. Let S be the set of UAV sub-tours $S = \{s_1, ..., s_k\}$. Let a UAV sub-tour $s_i \in S$ be denoted as a pattern $\{v_j, t_i, t_{i+1}, ..., t_p, v_j\}$ where $v_j \in V$ and $t_i \in T$ for $i = 1, ..., p$. Note that a UAV sub-tour $s_i \in S$ could have intermediate visits to the same $v_j \in V$, e.g. $\{v_j, t_i, v_j, t_{i+1}, v_j\}$, which means the UAV needs an intermediate charging stop on the UGV before to continue the exploration. Let g_{v_j,t_i} or g_{t_j,t_i} denote the travel cost as the distance travelled by the UAV to travel from the vertex $v_j \in V$ or the target $t_j \in T$ to $t_i \in T$. Hence, let F_{ugv} denote the UGV's travelling distance and let F_{uav} denote the UAV's travelling distance. Then, let F_{total} denote the total travelling distance of the simple UGV-UAV system. The ECU-CSURP goals are summarized in the following objective functions (3.1-3.3):

$$\text{minimize} \rightarrow F_{ugv} = \sum_{i=0}^{m-1} f_{v_{i,i+1}} + f_{v_{m,0}} \tag{3.1}$$

$$\text{minimize} \rightarrow F_{uav} = \sum_{\substack{j=0,\\ i=1}}^{m} \left[g_{v_j,t_i} + \left(\sum_{i=1}^{p-1} (g_{t_i,v_j} + g_{v_j,t_{i+1}}) * x_i + \right.\right.$$
$$\left.\left. + g_{t_{i,i+1}} * \overline{x_i} \right) + g_{t_p,v_j} \right] \tag{3.2}$$

$$\text{minimize} \rightarrow F_{total} = F_{ugv} + F_{uav} \tag{3.3}$$

Where x_i is a binary variable which takes a value 1 if there is an intermediate charging stop from the target $t_i \in T$ to the vertex $v_j \in V$, and 0 otherwise. The objective function 3.1 aims to minimize the UGV's travelling distance to accomplish its path. The objective function 3.2 aims to minimize the UAV's travelling distance to accomplish every sub-tour. And, finally, the objective function 3.3 formulates the travelling distance of the heterogeneous simple UGV-UAV system.

3.2 THE TERRA ALGORITHM FOR $\mathbb{R}^2$ EUCLIDEAN SPACES

The cooperative path planning algorithm implemented to solve the ECU-CSURP is formally named as cooperaTive ExploRation Routing Algorithm (TERRA) [164]. Figure 3.2 shows a graphical representation of an ECU-CSURP instance, displaying the main elements that TERRA will consider to minimize the objective functions 3.1-3.3.

TERRA exploits a particular cooperation synergy between the UGV and the UAV, which represents the interactions between the robot

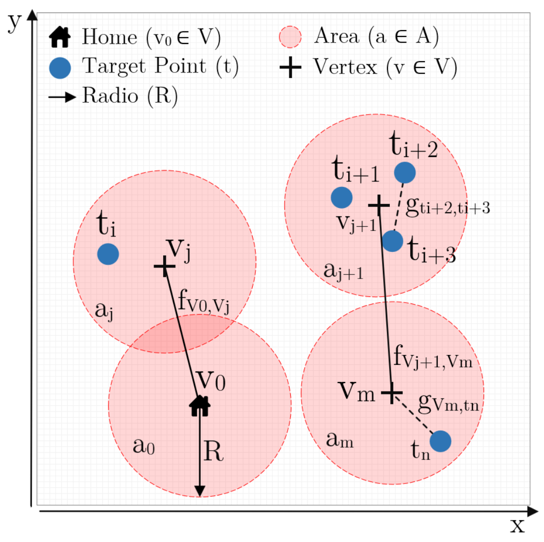


Figure 3.2: A TERRA representation of the ECU-CSURP instance shown in Figure 3.1. There are placed in a $\mathbb{R}^2$ Euclidean space (X,Y): a set of target points $(t_i, t_{i+1}, t_{i+2}, t_n \in T)$, a possible set of vertices $(v_j, v_{j+1}, v_m \in V)$, a set of areas $(a_0, a_j, a_{j+1}, a_m \in A)$ for each $v \in V$ and a home location $(V_0 \in V)$. The arrow is the farthest distance R the UAV can travel.

systems to overcome a common goal. Here, the UGV is a moving charging station which carries the UAV through charging stops from where the UAV can reach the target points. The UAV can recharge its battery on the on-board UGV charging station while the UGV is at a charging stop. Each charging stop is linked to a subset of target points, so on each subset, the UAV plans a route always considering the possibility to fly back to the UGV for recharging and avoid run out of energy in the next subset of target points. Once the UAV has achieved a subset of target points, it flies back to the UGV. Then, the UGV carries the UAV to the next planned charging stop to visit the associated subset of target points. The exploration problem is completed when every target point has been visited and both systems reach the home location.

Our algorithm is split up in an ordered sequence of five stages, and each stage has been properly addressed in the next five sub-sections respectively. The next sub-section or first stage presents a Voronoi search method to find a set of vertices aiming to minimize objective function 3.1. On the second stage, a method is depicted to reduce the cardinality of the set of vertices an so, minimize the objective function 3.1. The third stage is based on reducing the $\mathbb{R}^2$ Euclidean distance among the vertices and so, keep minimizing the objective function 3.1. The fourth stage[3] computes the sTSP to find a sub-optimal path for the UGV and minimize the objective function 3.1. Finally, the fifth stage

3 The sTSP is the symmetric TSP explained in subsection 2.1.3.

computes the sTSP to find a sub-optimal path for the UAV on every sub-tour and minimize the objective functions 3.2 and 3.3.

3.2.1 *A Voronoi's search*

The TERRA's first stage implements a Voronoi's search method for placing intermediate charging stops for the UGV, so the UAV can reach the target points without running out of energy. The computational method follows a similar approach to the Preparata and Shamos's method [40] (see subsection 2.1.1) to compute the NNS, by partitioning the space into regions computing Voronoi tessellations (see Figure 3.3). Nevertheless, the goal in our search method is to find a set of Voronoi's vertices $v_j \in V$, so that $g_{v_0,t_i} = f_{v_0,v_j} + g_{v_j,t_i}$ and whose areas $a_j \in A$ cover the whole set of target points.

Charging stops are also named as vertices.

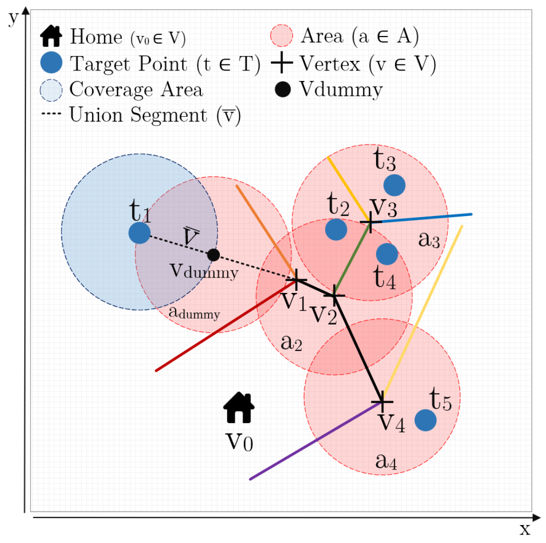


Figure 3.3: First stage of TERRA for the ECU-CSURP instance shown in Figure 3.1. Every target point t_i is covered by at least one vertex v_j. Note that the home location v_0 is considered as an input, but no area encompasses it because it does not cover any target point.

The pseudo code of the Voronoi's search method implemented in TERRA is shown in Algorithm 2. The inputs are the set of target points T and the farthest distance R the UAV can travel. In the following we provide a detailed description of the functions that form this method:

- *Voronoi* (line 17): is the *voronoi function*[4] integrated in the MATLAB computational geometry package. It computes a Voronoi tessellation with the Voronoi points ($V_{vpoints}$) given as input. The output is a set of vertices which represent the called Voronoi vertices ($V_{vertices}$) (v_1,v_2, v_3 and v_4 in Figure 3.3).

4 https://es.mathworks.com/help/matlab/ref/voronoi.html

- *NearestVertex* (line 29): it computes the distance between each $t \in T_{nc}$ and every $v \in V_{vertices}$. Then, it gives as output the nearest v of each t.

- *UnionSegment* (line 14): it computes the union segment ($\bar{v}$ in Figure 3.3) formed by the two vertices in $V_{vpoints}$ (only when it complies the equality in line 13). A vertex corresponds to the latest remaining target point in T_{nc} and the other to its nearest vertex in V_{nears} (union in line 31).

- *JunctionPoint* (line 15): it computes the junction point between the Linear Equation traced by the segment ($\bar{v}$ in Figure 3.3) and the Equation of the Circle of the latest target point (t_1 in Figure 3.3) following the Pythagorean Theorem. Thus, it solves the following system of equations:

$$\left.\begin{aligned} \frac{X - x_1}{x_2 - x_1} = \frac{Y - y_1}{y_2 - y_1} \\ (X - x_1)^2 - (Y - y_1)^2 = R^2 \end{aligned}\right\} \tag{3.4}$$

where (x_1, y_1) and (x_2, y_2) are the Cartesian Coordinates of the latest remaining target point in T_{nc} and its nearest vertex in V_{nears} (t_1 and v_1 in Figure 3.3). It returns the junction point (X, Y) as a false Voronoi vertex (V_{dummy} in Figure 3.3) required to cover the target point.

- *ComputeAreas* (line 33): it computes the set of areas $\{a_0, ..., a_m\} \in A$ where $a_j = \{t_1, ..., t_n\}$.

The algorithm performs iterative Voronoi tessellations covering the target points in T_{nc}. Sometimes, the algorithm detects that there are not enough $V_{vpoints}$ (condition in line 13) to compute a Voronoi tessellation, i. e., there is only one isolated target point remaining to be covered and its nearest vertex. In this situation, the algorithm generates a false Voronoi vertex which guarantees the coverage of the isolated target point (union in line 19).

The outputs are a set of Voronoi's vertices (also called as vertices or charging stops) V (Equation 3.5) and their areas A containing the set of target points (Equation 3.6). At this point, every target point in T has been covered by A.

$$\text{Voronoi's Search(T,R)} \rightarrow \{v_0, ..., v_m\} \in V,\ \{a_0, ..., a_m\} \in A \tag{3.5}$$

$$a_j \in A \leftarrow \{t_1, ..., t_n\} \tag{3.6}$$

Algorithm 2 Voronoi's search method

```
1: Procedure VoronoiSearch(T, R)
2: {$T_{nc}$: set of target points not covered yet}
3: {$V_{nears}$: set of nearest vertices for every $t \in T_{nc}$}
4: {$V_{vertices}$: set of Voronoi vertices }
5: {$V_{vpoints}$: set of Voronoi points }
6: {$v_{dummy}$: false Voronoi vertex to cover an isolated target point }
7: {$s_e$: union segment between a target point and a vertex}
8: $T_{nc} \leftarrow T$
9: $V, V_{vertices}, V_{nears} \leftarrow \emptyset$
10: $V_{vpoints} \leftarrow T_{nc}$
11: while $T_{nc} \neq \emptyset$ do
12:   $v_{dummy} \leftarrow \emptyset$
13:   if $|V_{vpoints}| == 2$ then
14:     $s_e \leftarrow UnionSegment(V_{vpoints})$
15:     $v_{dummy} \leftarrow JunctionPoint(s_e, R, T_{nc})$
16:   else
17:     $V_{vertices} \leftarrow Voronoi(V_{vpoints})$
18:   end if
19:   $V_{vertices} \leftarrow V_{vertices} \bigcup v_{dummy}$
20:   for $t \in T_{nc}$ do
21:     for $v \in V_{vertices}$ do
22:       if $distance(t, v) < R$ then
23:         $T_{nc} \leftarrow T_{nc} \setminus t$
24:         $V \leftarrow V \bigcup v$
25:       end if
26:     end for
27:   end for
28:   for $t \in T_{nc}$ do
29:     $V_{nears} \leftarrow V_{nears} \bigcup NearestVertex(t, V_{vertices})$
30:   end for
31:   $V_{vpoints} = T_{nc} \bigcup V_{nears}$
32: end while
33: $A \leftarrow ComputeAreas(V, R)$
34: return $V, A$
35: End procedure
```

3.2.2 *A combinatorial optimization algorithm*

The second stage aims to find a minimum set of vertices whose areas guarantee full covering of the set of target points (see Figure 3.4). In computer science this is called the HSP, which is a generalization of the SCP commented in subsection 2.1.2. It is modelled as a bipartite graph where the vertices $v_j \in V$ are represented on the left side, the universe is represented by the target points $t_i \in T$ on the right side,

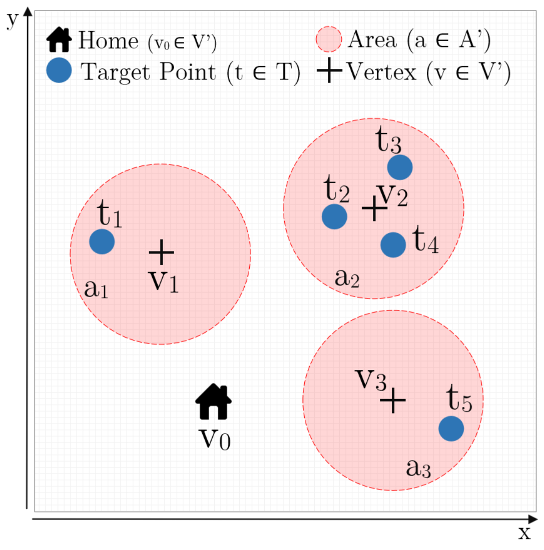


Figure 3.4: TERRA's second stage for the ECU-CSURP instance shown in Figure 3.1. The minimum set of vertices V' has been found to cover all the target points $t_i \in T$.

and the vertices $a_j \in A$ representing the inclusion of elements in sets. The task is to find a minimum cardinality subset of $v_j \in V$ whose $a_j \in A$ cover every $t_i \in T$.

The inputs are the vertices in V and the areas in A including the set of target points T. Figure 3.3 shows how TERRA realizes that v_2 (whose area a_2 covers $\{t_2, t_4\}$) can be deleted from the set of vertices V because v_3 already covers $\{t_2, t_3, t_4\}$, and thus, the cardinality is optimized, as Figure 3.4 shows. Therefore, the outputs are a minimum set of vertices V' (Equation 3.7) and their respective areas A' (Equation 3.8).

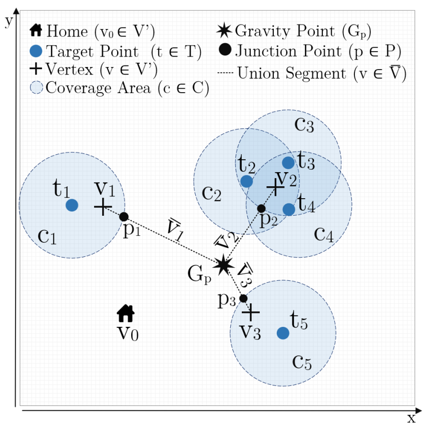
$$\text{HittingSet(V,A)} \rightarrow \{v_0, ..., v_m\} \in V',\ \{a_0, ..., a_m\} \in A' \quad (3.7)$$

$$a_j \in A' \leftarrow \{t_1, ..., t_n\} \quad (3.8)$$

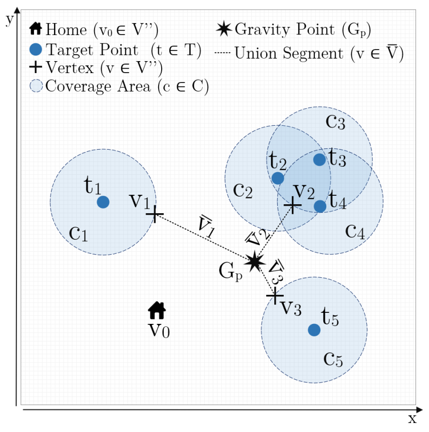
3.2.3 *A gravitational optimization algorithm*

The third stage aims to minimize F_{ugv} through the F_{uav} increment due to the following assumption: the UGV's motion speed is expected to be much slower than the UAV's motion speed in general explorations, i. e., $V_{ugv} << V_{uav}$. Consequently, F_{ugv} minimization will have higher impact than F_{uav} from a time perspective. Thus, given the areas $a_j \in A'$, the farthest distance R the UAV can travel, the vertices $v_j \in V'$ and a gravity point G_p, the objective is to find a set of vertices $p_j \in P$ where $d(p_j, G_p) < d(v_j, G_p)\ \forall\ p_j \in P$ and the constraint $g_{p_j,t_i} \leq R\ \forall\ t_i \in a_j \in A', p_j \in P$ is satisfied.

We developed a novel algorithm which places a gravity point G_p in the exploration area, and then, attracts the vertices $v_j \in V'$ to it by creating new vertices $p_j \in P$. The inputs are the farthest distance R



(a) Computing $p_j \in P$ with the gravitational algorithm.



(b) Replacing $p_j \in P$ with $v_j \in V''$ with the gravitational algorithm.

Figure 3.5: TERRA's third stage for the ECU-CSURP instance shown in Figure 3.1. In this instance, the solution selected is the one provided by the mean center (MC). The coverage areas $c \in C$ represent the maximum distance to replace each vertex v_j with p_j so that, the following constraint is satisfied: $g_{p_j,t_i} \leq R \, \forall \, t_i \in a_j \in A', p_j \in P$.

the UAV can travel, the gravity point G_p, and the set of vertices V' and areas A' solved in the previous stage. On each iteration, TERRA computes one solution without applying a gravity point and three gravity point solutions ($G_p = \{XC, HC, MC\}$), then, it chooses the best from all four. The three gravity points are defined as follows:

1. The mean center (XC): mean of the x-axis and y-axis of the target points.

2. The home location (HC): home location point of the ECU-CSURP instance.

3. The median center (MC): median of the x-axis and y-axis of the target points.

Algorithm 3 shows the pseudo code of the gravitational algorithm implemented in TERRA. The following functions summarize the algorithm's behaviour:

- *GetArea* (line 7): it obtains the target points covered by v.

- *UnionSegment* (line 8): it computes the union segment ($\bar{v}_j \in \bar{V}$ in Figure 3.5) between the vertex ($v_j \in V'$ in Figure 3.5a) and the G_p considered.

- *JunctionPoint* (line 10): it builds the system of equations defined in the first stage (Equation 4.39), where (x_1, y_1) and (x_2, y_2) are the Cartesian Coordinates of the target point t_c (coverage area $c \in C$ in Figure 3.5a) and its covering vertex v. It returns a

Algorithm 3 Gravitational Optimization Algorithm

```
1:  Procedure GravitationalOptimization($A'$, $R$, $V'$, $G_p$)
2:  {$a_v$: set of target points covered by $v$}
3:  {$s_e$: segment traced from $v$ to $G_p$}
4:  {$P$: set of candidate junction points}
5:  $V'' \leftarrow \emptyset$
6:  for $v \in V'$ do
7:      $a_v \leftarrow GetArea(A', v)$
8:      $s_e \leftarrow UnionSegment(v \bigcup G_p)$
9:      for $t_c \in a_v$ do
10:         $P \leftarrow P \bigcup JunctionPoint(s_e, R, t_c)$
11:     end for
12:     $V'' \leftarrow V'' \bigcup GetMin(P, G_p)$
13: end for
14: return $V''$
15: End procedure
```

candidate junction point ($p_j \in P$ in Figure 3.5a) to replace the vertex $v_j \in V$.

- *GetMin* (line 12): it computes the distance between G_p and every candidate junction point $p \in P$. Then, it gives as output the nearest candidate to replace the vertex ($v_j \in V''$ in Figure 3.5b).

The gravitational algorithm computes finite iterations until it finds a candidate junction point to replace every vertex. The output is a set of replaced vertices V'' (Equation 3.9).

$$\text{GravitionalAlgorithm(A',R,V'},G_p) \rightarrow \{v_0, ..., v_m\} \in V'' \tag{3.9}$$

3.2.4 *A genetic algorithm for the UGV's path*

The fourth stage aims to compute the shortest UGV's directed path in order to minimize F_{total}. This problem is the well-known TSP presented in subsection 2.1.3. Given a set of vertices $V'' \leftarrow \{v_0, ..., v_m\}$ and the distances between each pair of vertices f_{v_i,v_j} $\forall$ $(v_i, v_j \in V'')$, the task is to find the shortest possible route that visits each vertex and returns to the home location v_0. Particularly, we model the symmetric generalization sTSP, where the distance f_{v_i,v_j} between the vertex v_i and the vertex v_j follows $f_{v_i,v_j} = f_{v_j,v_i}$.

We developed a genetic algorithm which takes as input the set of vertices V'' computed in the previous stage. Our algorithm is built under three main evolutionary steps. The first step is a selection mechanism where a portion of the existing population is selected to breed a new generation. It applies the tournament selection method [248] to repeatedly select the best individual of a randomly chosen subset. The second step implements an elitist selection process in which the best individuals from the current population are carried over to the next, unaltered. The tournament and elitist selections methods ensure a generational process which keeps the population size constant on each generation. The third step produces the new generation from those selected in the tournament selection through the combination of the genetic operators: mutation and crossover. The mutation operator uses flipping, swapping and sliding techniques to create new chromosomes. The crossover operators applied are: Order Crossover, Cycle Crossover and Order Base Crossover. In order to properly adjust the genetic algorithm to ECU-CSURP instances, we performed a tuning experiment to select an appropriate parameter configuration (see section 3.4). The output is an ordered path (Equation 3.10) minimizing the objective function 3.1 (see Figure 3.6).

$$\text{GeneticAlgorithm(V'')} \rightarrow \{v_0, v_j, v_{j+1}, ..., v_m, v_0\} \in V''' \qquad (3.10)$$

3.2.5 *A search algorithm for the UAV's path*

The fifth stage aims to compute the shortest UAV's directed path among the multiple UAV sub-tours in order to minimize F_{total}. This problem involves multiple symmetric TSPs. Given a set of areas A', where each area $a_j \in A'$ is an unordered set of target points $\{t_1, ..., t_n\}$, the farthest distance R the UAV can travel, a set of vertices V'', and the distances g_{v_j,t_i} $\forall$ $v_j \in V'', t_i \in a_j \in A'$ and g_{t_j,t_i} $\forall$ $t_j, t_i \in a_j \in A'$; the objective is to find the multiple shortest sub-tours $s_i \in S$ which visits each target point $t_i \in a_j \in A'$ and returns to the linked charging stop $v_j \in V''$.

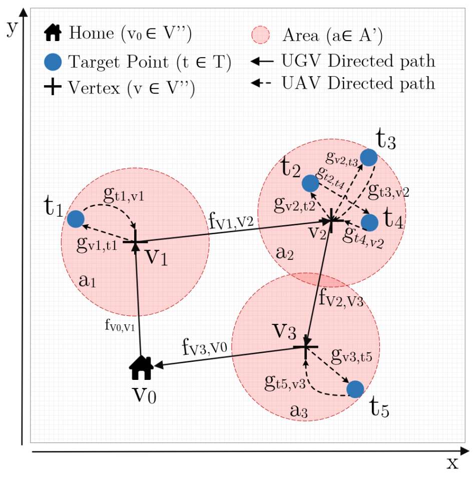


Figure 3.6: Fourth and fifth stage of TERRA for the ECU-CSURP instance shown in Figure 3.1. The fourth stage gives as output the UGV's directed path, and the fifth stage gives the UAV's directed path.

We developed a search algorithm similar to the A* algorithm [249]. The inputs are the set of areas A' computed in the second stage, the farthest distance R and the set of vertices V'' computed in third stage. As well as A*, it uses the evaluation function $f = g + h$, where the cost function g is the accumulated distance to reach a target point and the heuristic function h represents the remaining target points to visit.

Algorithm 4 The search algorithm

```
1:  Procedure searchRouting($A'$,$V''$,$R$)
2:  {$a_v$: set of target points covered by $v$}
3:  {$s$: a UAV sub-tour}
4:  {$open, closed$: list of nodes to be visited and already visited nodes}
5:  {$p_{start}$: starting node or charging stop of each sub-tour}
6:  {$p_{curr}$: current node of the search graph}
7:  {$p_{neighbours}$: neighbour nodes list of $p_{curr}$}
8:  for $v \in V''$ do
9:      $a_v \leftarrow GetArea(A', v)$
10:     $a_v \leftarrow a_v \bigcup v$
11:     $coord(p_{start}) \leftarrow v$
12:     $d(p_{start}), g(p_{start}) \leftarrow 0$
13:     $h(p_{start}) \leftarrow |a_v| - 1$
14:     $f(p_{start}) = g(p_{start}) + h(p_{start})$
15:     $parent(p_{start}) \leftarrow p_{start}$
16:     $closed, open \leftarrow \emptyset$
17:     $open.Push(p_{start})$
18:     while $open \neq \emptyset$ do
19:         $p_{curr} \leftarrow open.Pop()$
20:         if $coord(p_{curr}) == v$ and $h(p_{curr}) == 0$ then
21:             return $s \leftarrow GetSubTour(p_{curr})$
22:         end if
23:         $closed.Push(p_{curr})$
24:         $p_{neighbours} \leftarrow ExpandGraph(p_{curr}, p_{start}, R, a_v)$
25:         for $p_{nbr} \in p_{neighbours}$ do
26:             if $p_{nbr} \notin closed$ then
27:                 if $p_{nbr} \notin open$ then
28:                     $g(p_{nbr}) \leftarrow Inf$
29:                     $parent(p_{nbr}) \leftarrow \emptyset$
30:                 end if
31:                 $open \leftarrow UpdateNode(open, p_{curr}, p_{nbr})$
32:             end if
33:         end for
34:     end while
35:     return $S \leftarrow S \bigcup s$
36: end for
37: return $S$
38: End procedure
```

Algorithm 4 shows the pseudo code of the search algorithm. It computes finite sub-tours $s \in S$ for each $v \in V''$ looking for the shortest directed path. Each sub-tour denotes the starting charging stop v as the p_{start} node. A node is an object containing the following information: the Cartesian Coordinates (*coord*) of the target point in a_v or the vertex in V'', its accumulated distance from the last charging stop (d), its cost, heuristic and evaluation functions (g, h, t) and its parent node (*parent*). The following functions help to understand it:

- *getSubTour* (line 21): it recursively obtains the parent nodes of the search graph starting from p_{curr}.
- *ExpandGraph* (line 24, detailed in Algorithm 5): it integrates the UAV energy constraint R into the searching process.

Algorithm 5 Expanding the graph looking for neighbours

```
1:  **Procedure ExpandGraph**($p_{curr}, p_{start}, R, a_v$)
2:  {$p_{nbr}$: $p_{curr}$ *neighbour node expanded in the graph*}
3:  {*n: candidate to neighbour node*}
4:  $p_{neighbours} \leftarrow \emptyset$
5:  **for** $n \in a_v$ **do**
6:    **if** $coord(p_{curr}) \neq n$ **and not** $relative(p_{curr}, n)$ **then**
7:      **if** $coord(p_{curr}) == coord(p_{start})$ **then**
8:        $d \leftarrow distance(coord(p_{curr}), n)$
9:      **else**
10:       $d \leftarrow d(p_{curr}) + distance(coord(p_{curr}), n)$
11:     **end if**
12:     $d_{start} \leftarrow distance(coord(p_{start}), n)$
13:     **if** $R * 2 >= d + d_{start}$ **then**
14:       **if** $n == coord(p_{start})$ **then**
15:         $h \leftarrow h(p_{curr})$
16:       **else**
17:         $h \leftarrow h(p_{curr}) - 1$
18:       **end if**
19:       $coord(p_{nbr}) \leftarrow n$
20:       $parent(p_{nbr}) \leftarrow p_{curr}$
21:       $d(p_{nbr}) \leftarrow d$
22:       $h(p_{nbr}) \leftarrow h$
23:       $g(p_{nbr}) \leftarrow p_{curr}.g + distance(coord(p_{curr}), n)$
24:       $f(p_{nbr}) \leftarrow g(p_{nbr}) + h(p_{nbr})$
25:       $p_{neighbours}.Push(p_{nbr})$
26:     **end if**
27:   **end if**
28: **end for**
29: **return** $p_{neighbours}$
30: **End procedure**
```

At first, it checks if $n \in a_v$ has already been visited (a node can be visited only once in a route) (line 6). Then, if $coord(p_{curr})$ is equal to $coord(p_{start})$ (line 7), it means that the route does an intermediate charging stop and it is starting again to visit the remaining target points, i.e., the accumulated distance from the last charging stop (d) has to be equal to the distance required to visit the neighbour n. If not, it accumulates the travelling distance to the neighbour n. Then, it checks that the sum of the distance required to visit n plus the distance required from n to return to the charging station p_{start} ($d + d_{start}$) is less than the maximum distance the UAV can travel ($d_{max} = R * 2$) ensuring the return to the charging station (line 13). If not, n is not a valid neighbour and it is not included into the search graph because the UAV would run out of energy following that route. Otherwise, the node n is included into the neighbours list $p_{neighbours}$. Note that p_{start} does not compute as a real target point in h (line 14).

- *UpdateNode* (line 31, detailed in Algorithm 6): it checks if the new distance ($d + g(p_{nbr})$) computed to reach p_{nbr} node from the p_{curr} parent is shorter than the distance ($g(p_{nbr})$) previously computed to reach p_{nbr} from another parent node (line 4). Thereupon, it updates the travel cost g, *parent* and the evaluation function f, and updates the node in the *open* nodes list.

Algorithm 6 Updating the selected node

```
1:  Procedure UpdateNode($open, p_{curr}, p_{nbr}$)
2:  {$p_{nbr}$: $p_{curr}$ neighbour node}
3:  $d \leftarrow distance(coord(p_{curr}), coord(p_{nbr}))$
4:  if $g(p_{curr}) + d < g(p_{nbr})$ then
5:      $g(p_{nbr}) \leftarrow g(p_{curr}) + d$
6:      $parent(p_{nbr}) = p_{curr}$
7:      if $p_{nbr} \in open$ then
8:          $open.Remove(p_{nbr})$
9:      end if
10:     $f(p_{nbr}) \leftarrow g(p_{nbr}) + h(p_{nbr})$
11:     $open.Push(p_{nbr})$
12: end if
13: return $open$
14: End procedure
```

For each $v \in V''$, our search algorithm returns an ordered set of locations $s \in S$ denoting a UAV sub-tour where the UAV's travelling distance has been minimized as Equation 3.11 shows.

$$F_{s \in S}(v_j \in V'', t_i \in a_j \in A') =$$

$$g_{v_j,t_i} + \left(\sum_{i=1}^{p-1} g_{t_i,v_j} * x_i + g_{v_j,t_{i+1}} * x_i + g_{t_i,t_{i+1}} * \overline{x_i} \right) + g_{t_p,v_j} \quad (3.11)$$

The output is a set of UAV sub-tours S (Equation 3.12) minimizing the objective function 3.2 as in Equation 3.13 (see Figure 3.6).

$$\text{SearchAlgorithm(A',V'',R)} \rightarrow \{s_1, ..., s_k\} \in S \quad (3.12)$$

$$F_{uav} = \sum_{i=1}^{k} F_{s_i \in S} \quad (3.13)$$

Finally, once F_{ugv} and F_{uav} have been minimized, the objective function 3.3 is minimized and the ECU-CSURP is solved.

3.3 EXTENDING TO $\mathbb{R}^3$ EUCLIDEAN SPACES

The TERRA algorithm presented in section 3.2 is designed for $\mathbb{R}^2$ exploration problems, which means that only flat terrains can be considered. Also, it does not consider terrain obstacles, so, in practice, the exploration area has to be a complete open and flat terrain. This algorithm can be admissible for some scenarios in these ideal terrains e. g., some industrial or urban areas. However, it is not suitable for scenarios in cumbersome terrains e. g., high-hill or uneven areas, which can also have terrains features, such as rocks or mountain ridges.

As we stated at the beginning of this chapter, our objective is to design a cooperative simple UGV-UAV path planning algorithm for a broad range of high-level explorations. Thus, we describe the extension of the exploration problem and TERRA to $\mathbb{R}^3$ Euclidean spaces in the present section. Next sub-section describes the definition changes of the problem formulation, and subsection 3.3.2 the TERRA corresponding enhancements.

3.3.1 *The exploration problem in* $\mathbb{R}^3$

The ECU-CSURP stated in section 3.1 defines in its first constraint that the exploration area has to be modelled as a $\mathbb{R}^2$ Euclidean space. Extending the ECU-CSURP to $\mathbb{R}^3$ implies extending this constraint to $\mathbb{R}^3$ Euclidean spaces. In particular, we constrain the $\mathbb{R}^3$ area to be represented as a Digital Terrain Model (DTM), which is a well-used 3D computer graphic representation of $\mathbb{R}^3$ Euclidean spaces. A DTM represents the bare ground surface without any objects like trees or buildings. Figure 3.7 shows an ECU-CSURP instance in a DTM of the Mars surface. Specifically, it shows the central uplift of

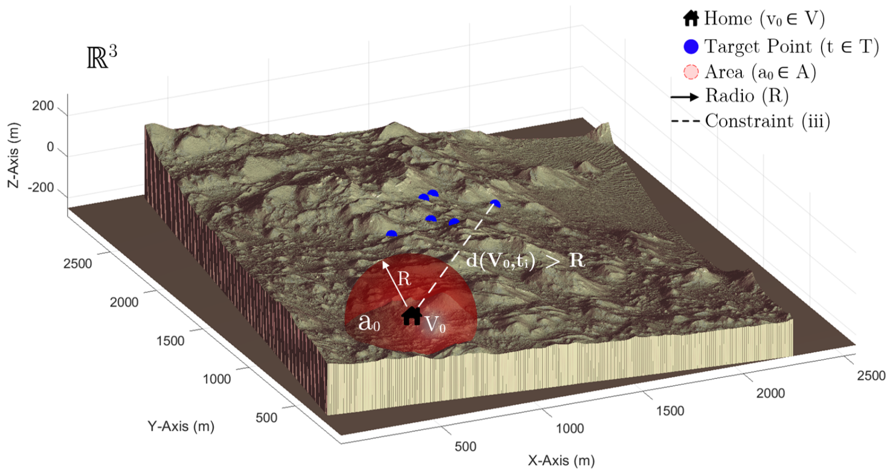


Figure 3.7: An ECU-CSURP instance where a set of target points $t \in T$ and a home location $v_0 \in V$ are distributed around a $\mathbb{R}^3$ Euclidean space. This Euclidean space is represented as a real Mars DTM. R denotes the farthest distance the UAV can travel considering a return flight to the take-off point. The red area a_0 is the area covered by the UAV from the home location. The distance constraint is also satisfied with $d(V_0, t_i) > R$.

a 30-Km diameter crater in Noachis Terra[5], captured by the High Resolution Imaging Science Experiment (HiRISE) on board the Mars Reconnaissance Orbiter. There, we can appreciate that the ECU-CSURP constraints are met: (i) the exploration area is modelled as a $\mathbb{R}^3$ area or DTM, (ii) a set of target points $t \in T$ to be reached, (iii) a distance constraint such that $d(V_0, t_i) > R$, (iv) a heterogeneous simple UGV-UAV system, (v) a UAV energy constraint modelled as the coverage area a_0, and (vi) a home location $V_0 \in V$ where the UGV-UAV systems starts and ends the exploration.

We use the DTM geometric formulation of Muñoz et al. [250], to compute the objective functions 3.1-3.3 of ECU-CSURP. This formulation define how to compute three-dimensional distances and the slope of each node in order to obtain safer routes. They define a method to compute the approximate distance travelled by a robot through adjacent (or not adjacent) three-dimensional nodes. For that, they interpolate the elevation of nodes that are inside of the rectangular cell. Here, a node is a coordinate tuple (x_p, y_p, z_p), where z_p is the elevation obtained from the DTM. We use this notation to represent the target points, charging stops, UGV's path and UAV's path in TERRA.

Muñoz et al. [250] model a DTM as a grid of rectangular cells, where each cell is formed by four adjacent nodes in the map. These four nodes along with a central point of the cell form four triangles. The slope of each triangular plane determines the slope of each node at the rectangular cell. This model expresses this computed slope and

5 Its identification in the HiRISE database is: DTEED-030808-1535- 031230-1535-A01.

other terrain characteristics as a numerical value. This value indicates the cost (estimated effort) required to cross an area of the map. We use this cost to identify the following two kind of nodes:

- Legitimate nodes: those whose can be crossed by the UGV because the cost its lower than a given threshold, and represent a feasible option for a charging stop.
- Illegitimate nodes: those whose cannot be crossed by the UGV because the cost its higher than a given threshold.

3.3.2 *Updating TERRA for* $\mathbb{R}^3$

Once we extended the ECU-CSURP to $\mathbb{R}^3$ Euclidean spaces, we updated TERRA to solve $\mathbb{R}^3$ problem instances [251]. On one hand, the updates we devised for the five stages of $\mathbb{R}^2$ TERRA, could be summarized by replacing the $\mathbb{R}^2$ coordinate tuple (x_p, y_p) by the $\mathbb{R}^3$ tuple (x_p, y_p, z_p) to every stage of the algorithm. Note that the stages based on combinatorial optimization problems (the second, fourth and fifth stages) do not care about the dimensionality of the exploration area. However, the Voronoi's search and the gravitational optimization stages required some updates. On the other hand, updating TERRA for $\mathbb{R}^3$ problem instances is strictly related to compute three-dimensional paths for the heterogeneous simple UGV-UAV system. Therefore, we integrated the 3Dana algorithm [250] to compute the UGV's three-dimensional path, whereas for the UAV's path, we defined a constant altitude. In the following, we describe the changes of the the Voronoi's search and the gravitational optimization stages to upgrade to $\mathbb{R}^3$ scenarios, and the new stage to compute the three-dimensional UGV's path.

As the TERRA's first stage, our Voronoi's search method looks for placing intermediate charging stops in the UGV's path, so the UAV can reach the target points without running out of energy. In $\mathbb{R}^2$ Euclidean spaces, the map is an open (without obstacles) and flat terrain, so every point is a legit node (also called as legit vertex) to be a charging stop, because the UGV can cross every point without problems. Nevertheless, as we mentioned in the previous section, a point may not be legit to be crossed by an UGV in DTMs. Therefore, we updated our Voronoi's search method as shows the pseudo code in Algorithm 7. This algorithm has the following three main changes referring to Algorithm 2:

1. The computed Voronoi vertices in the *Voronoi* function (line 19) are formed by legit vertices (V_{legit}), unlegit vertices ($V_{unlegit}$) and vertices which are out of the map boundaries. Then, this distinction is mandatory in order to select legitimate vertices that cover the target points, and the nearest vertices of each target point not covered yet (line 33).

2. The *distance3D* function (line 26) computes the three-dimensional distance between a target point and a legit vertex using Pythagoras.

Algorithm 7 3D Voronoi's search method

```
1:  Procedure Voronoi3DSearch($T$, $R$, $Map$)
2:  {Map: cost map of the DTM}
3:  {$T_{nc}$: set of target points not covered yet}
4:  {$V_{nears}$: set of nearest vertices for every $t \in T_{nc}$}
5:  {$V_{vertices}$: set of Voronoi vertices }
6:  {$V_{legit}$: set of Voronoi legit vertices }
7:  {$V_{unlegit}$: set of Voronoi unlegit vertices }
8:  {$V_{vpoints}$: set of Voronoi points }
9:  {$v_{dummy}$: false Voronoi vertex to cover an isolated target point }
10: {$s_e$: union segment between a target point and a vertex}
11: $T_{nc} \leftarrow T$
12: $V_{vpoints} \leftarrow T_{nc}$
13: $V, V_{vertices}, V_{nears}, V_{legit}, V_{unlegit} \leftarrow \emptyset$
14: while $T_{nc} \neq \emptyset$ do
15:     $v_{dummy} \leftarrow \emptyset$
16:     if $|V_{vpoints}| == 2$ then
17:         $v_{dummy} \leftarrow greenZoneSearch(V_{vpoints}, R, Map)$
18:     else
19:         $V_{vertices} \leftarrow Voronoi(V_{vpoints})$
20:         $V_{legit} \leftarrow getLegits(V_{vertices}, Map)$
21:         $V_{unlegit} \leftarrow getNoLegits(V_{vertices}, Map)$
22:     end if
23:     $V_{legit} \leftarrow V_{legit} \bigcup v_{dummy}$
24:     for $t \in T_{nc}$ do
25:         for $v \in V_{legit}$ do
26:             if $distance3D(t, v) < R$ then
27:                 $T_{nc} \leftarrow T_{nc} \setminus t$
28:                 $V \leftarrow V \bigcup v$
29:             end if
30:         end for
31:     end for
32:     for $t \in T_{nc}$ do
33:         $V_{nears} \leftarrow V_{nears} \bigcup NearestVertex(t, V_{legit}, V_{unlegit})$
34:     end for
35:     $V_{vpoints} = T_{nc} \bigcup V_{nears}$
36: end while
37: $A \leftarrow ComputeAreas(V, R)$
38: return $V, A$
39: End procedure
```

3. The *greenZoneSearch* function (line 17) computes the false Voronoi vertex (v_{dummy}) to cover an isolated target point. As well as in Algorithm 2, it happens when the *Voronoi* function cannot compute a feasible Voronoi tessellation with only two vertices. The *greenZoneSearch* function roams around the target point in a spiral mode from the outbounds delimited by R, aiming to find a legit vertex inside the boundaries of the target point.

The introduction of legitimate and illegitimate vertices does not only impact to the Voronoi's search method, but also to the gravitational optimization algorithm. In this way, Algorithm 3 is updated to detect if the computed junction point is a legit or an unlegit vertex. That is, the new algorithm works in a same way but computing the corresponding $\mathbb{R}^3$ Euclidean functions. Furthermore, the *JunctionPoint* function (line 10 in Algorithm 3), includes an additional step where it checks if the junction point is a legit or an unlegit vertex. In case of a legit vertex, it keeps the execution. Otherwise, it searches a legit vertex into the union segment ($\bar{v}_j \in \bar{V}$ in Figure 3.5) between the vertex ($v_j \in V'$ in Figure 3.5a) and the G_p considered.

Nevertheless, as we introduced in this sub-section, computing three-dimensional UGV's paths represents a significant enhancement of the TERRA algorithm to solve $\mathbb{R}^3$ problem instances. However, we do not compute the UAV's three-dimensional path as well as other physical constraints, such as wind speed or atmospheric density, for two reasons: first, some of them imply an explosion in the problem complexity and, second, dynamic constraints (as wind speed) cannot easily modelled neither predicted to be exploited in off-line planning (they are better suitable for on-line adaptation of the plan during execution). Instead, we define a constant flight altitude for the UAV.

For the UGV's three-dimensional path, we introduce an additional stage after computing the TSP for the two-dimensional UGV's path (fourth stage). This new stage is the 3Dana algorithm [250], a path planning algorithm developed to obtain safer routes based on heuristic search over a DTM and/or a traversability cost map. The 3Dana algorithm generates long term paths exploiting a DTM geometric formulation without requiring a mechanical model of the robot. Furthermore, it provides the capability of combining both traversability cost maps and DTMs, so the paths generated are safer than the ones obtained by exploiting representations combined into a single cost map.

3.4 EXPERIMENTAL EVALUATION

In this section we present the TERRA experimental evaluation for different randomly generated ECU-CSURP instances. Firstly, we assess the TERRA performance to provide an accurate algorithm characterization in $\mathbb{R}^2$ Euclidean spaces. And secondly, we study the additional implications of executing TERRA in $\mathbb{R}^3$ Euclidean spaces. Appendix A shows the map generation algorithm used to create random ECU-CSURP ($\mathbb{R}^2$ and $\mathbb{R}^3$) instances, and the parameter tuning performed to set up the genetic algorithm for a proper evaluation.

The algorithm has been implemented in MATLAB, and all the experiments were carried out on a 2.6 GHz Intel Core i7 with 16 GB of RAM under Windows 10. The results of the experiments are publicly available on GitHub[6]. Additionally, Appendix B shows the TERRA computational results in instances generated using the TSP library TSPLib [252], and some additional statistical tests.

3.4.1 *Characterizing TERRA in* $\mathbb{R}^2$

The assessment of the algorithm has been focused in the trade-off between the UGV's travelling distance (objective function 3.1) and the UAV's travelling distance (objective function 3.2), and their impact in the total travelling distance (objective function 3.3). We began by evaluating the performance looking at the main parameter fluctuations of the random map generator (see Appendix A). The main parameters can be summarized as follows:

- N: number of target points.
- R: farthest distance the UAV can travel in km.
- δ: number of clusters, where a cluster is a group of closely located target points.

We assumed N as a constant parameter during the experimental evaluation because of its linear correlation with the objective function 3.3, i. e., a constant N increment will approximately get a constant F_{total} increment. Therefore, we focused the TERRA assessment on the R and δ parameters, and we performed two experiments: one experiment to analyse the correlation between R and the objective function 3.3, and other experiment to analyse the correlation between δ and the objective function 3.3.

3.4.1.1 *Overall TERRA performance*

The objective is to analyse the performance over instances with different R and δ values. As we mentioned above, N has been kept constant being $N = 16$.

The experiment setting is based on nine parameters configurations combining three different values of $R = \{1, 3, 9\}$ and $\delta = \{2, 4, 8\}$. Each parameter configuration will be tested over five hundred random ECU-CSURP instances. On each instance, TERRA will select the best of

6 https://github.com/FRopero/TERRA_Experiments

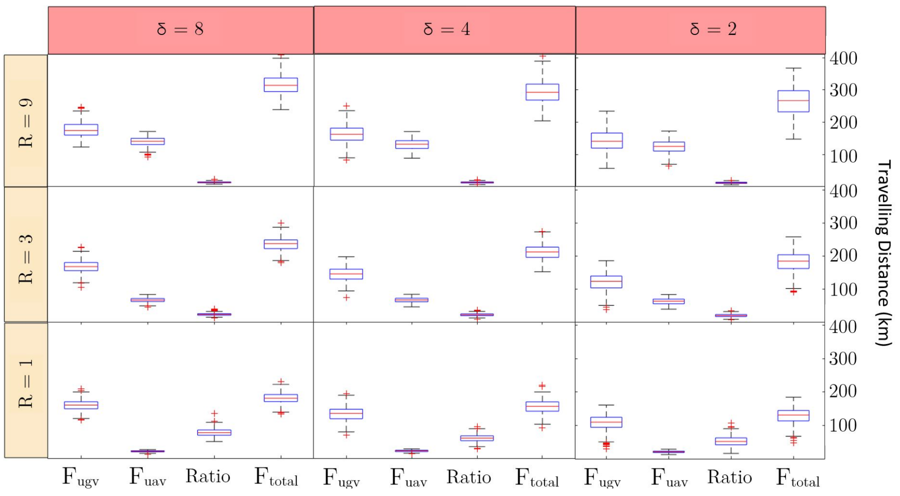


Figure 3.8: Results for the TERRA execution in nine different δ and R configurations. Each configuration has been evaluated over five hundred randomly generated maps. We can assert that there is a direct correlation between R and F_{total}, and δ and F_{total}.

the four gravity point solutions in the third stage. Figure 3.8 shows the experiment results in a box-plot matrix. δ (top x-axis) is the number of clusters. R (left y-axis) is the farthest distance R the UAV can travel in km. The parameters to evaluate (bottom x-axis) are: F_{ugv}, F_{uav}, $Ratio = F_{ugv}/F_{uav}$ and F_{total}. The parameter values (right y-axis) are displayed as the travelling distance in km.

Firstly, we observe that the increment of R generates a significant increment of F_{uav} mainly due to the gravitational optimization algorithm. That is, the higher R, the higher is the attraction of the gravity point ($d(v_j, G_p)$ in third stage) and the higher is the reduction of the distance among the vertices and the gravity point ($d(p_j, G_p)$ in third stage), which conducts to a F_{uav} increment and a F_{ugv} decrement. Despite of this F_{ugv} decrement, the R increment building problem instances leads to a F_{ugv} increment due to the dispersion of the target points locations on each cluster (Appendix A shows that the standard deviation $\sigma = a * R$, where $a \in \{0.5, 2\}$). That is, the higher R, the more scatter are the target points on each cluster, which in turns conducts to TERRA to require more vertices to cover the cluster and so, F_{ugv} is incremented. This F_{ugv} increment slightly dominates over the F_{ugv} decrement because of the gravitational optimization algorithm. Due to $F_{total} = F_{ugv} + F_{uav}$, we can assert the following Lemma 3.4.1:

Lemma 3.4.1. There is a direct correlation between R and F_{total}.

Additionally, we performed One Way ANOVA [253] tests to demonstrate the statistical significance of the Lemma 3.4.1. Please, refer to Appendix B to observe that Table B.1 shows that every p-value is lower than the significance level $\alpha = 0.05$, which demonstrates the statistical significance of the Lemma 3.4.1.

Secondly, we observe that a δ decrement results in a F_{ugv} decrement. That is, the lower δ, the lower is the number of vertices and so, F_{ugv}. This effect dominates over the gravitational optimization algorithm and it explains the F_{ugv} decrement. Also, we can observe that F_{uav} decreases with the δ decrement. This is because the search algorithm computing the UAV's path. Due to $F_{total} = F_{ugv} + F_{uav}$, we can assert the following Lemma 3.4.2:

Lemma 3.4.2. There is a direct correlation between δ and F_{total}.

As well, we driven One Way ANOVA tests to demonstrate the statistical significance of the above Lemma 3.4.2. Please, refer to Appendix B to observe that Table B.2 shows that every p-value is lower than the significance level $\alpha = 0.05$, which also demonstrates the statistical significance of the Lemma 3.4.2.

3.4.1.2 *Analysing gravitational effects*

The objective is to assess Lemma 3.4.1 by evaluating the gravitational optimization algorithm impact in the four computed solutions, i. e., the three gravitational solutions with three different gravity points and the solution without applying a gravity point.

The experimental setting is based on four parameters configurations in which $R = \{2, 4, 8, 16\}$. The rest of key parameters have been kept constant in the experiment ($\delta = 1$ and $N = 6$). Each configuration has been tested over five hundred random problem instances. Figure 3.9 shows the experiment results split up into six plots. The top and bottom x-axis is the farthest distance R the UAV can travel in km. The top y-axis is the distance travelled in km. The bottom y-axis is the time taken to cover the distance in hours. Each plot shows the three TERRA solutions with the three gravity points and the TERRA solution without a gravity point.

On the one hand, the three top plots represent the results of the difference between the travelling distance of the solution without applying the gravitational optimization algorithm f_{ugv}, f_{uav} and f_{total}, and the travelling distance of the three gravity solutions $f(g)_{ugv}$, $f(g)_{uav}$, $f(g)_{total}$ where $g \in G_p$ is the gravity point. Let $G_p = \{XC, HC, MC\}$ be the set of gravity points (see their definitions in subsection 3.2.3). Then, the travelling distance deviation λ is computed as follows:

$$\lambda_{ugv} = f_{ugv} - f(g)_{ugv} \forall g \in G_p \tag{3.14}$$

$$\lambda_{uav} = f_{uav} - f(g)_{uav} \forall g \in G_p \tag{3.15}$$

$$\lambda_{total} = f_{total} - f(g)_{total} \forall g \in G_p \tag{3.16}$$

On the other hand, the three bottom plots represent a theoretical computation (there are not considered real-world variables such as

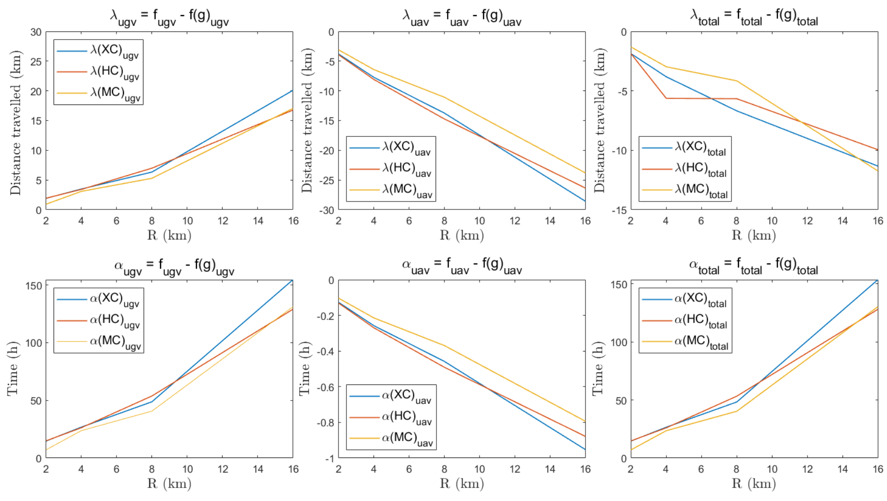


Figure 3.9: Results for the TERRA execution in four different parameters configurations with $R = \{2, 4, 8, 16\}$, $\delta = 1$ and $N = 6$. Each configuration has been evaluated over five hundred randomly generated problem instances. We conclude that $f(g)_{ugv}$ optimization will be more significant than $f(g)_{uav}$ optimization from a time perspective.

the wind speed or terrain slope) of the time taken by the heterogeneous simple UGV-UAV system to cover the distances F_{ugv} and F_{uav}, respectively. Thus, we need to add to TERRA the motion speed parameters. We follow the assumption commented at the beginning of subsection 3.2.3 ($V_{ugv} << V_{uav}$), denoting $V_{ugv} = 0.13km/h$ (maximum speed of the Mars Science Laboratory) as the UGV's motion speed, and $V_{uav} = 30km/h$ (standard value) as the UAV's motion speed. Also, let $t_{ugv} = f_{ugv}/V_{ugv}$, $t_{uav} = f_{uav}/V_{uav}$ and $t_{total} = t_{ugv} + t_{uav}$ denote the time taken to accomplish the mission without computing a gravity point. Let $t(g)_{ugv}$, $t(g)_{uav}$ and $t(g)_{total}$ denote the time taken to accomplish the mission computing a gravity point $g \in G_p$. Then, the total time deviation α is computed for each gravity point as follows:

$$\alpha_{ugv} = t_{ugv} - t(g)_{ugv} \forall g \in G_p \tag{3.17}$$
$$\alpha_{uav} = t_{uav} - t(g)_{uav} \forall g \in G_p \tag{3.18}$$
$$\alpha_{total} = t_{total} - t(g)_{total} \forall g \in G_p \tag{3.19}$$

From Figure 3.9, we can highlight three evidences. First, λ_{ugv} and α_{ugv} plots are in positive outcomes (top and bottom left plots). We can observe that the three gravity point solutions $f(g)_{ugv} \forall g \in G_p$ always improve the results against the solution without applying any gravity point f_{ugv}, i.e., $f(g)_{ugv} < f_{ugv}$. Then, the higher R, the lower is the distance among the junction points and the vertices ($d(p_j, G_p)$ in Figure 3.5), and so, the greater is the efficiency of the gravitational optimization algorithm.

Second, λ_{uav} and α_{uav} plots are in negative outcomes (top and bottom middle plots). Here, we can observe the main drawback in the trade-off between the UGV's and UAV's travelling distance. That is, when $\delta = 1$, the gravitational optimization algorithm computes the maximum F_{ugv} reduction by maximizing F_{uav}, as can be observed in Figure 3.5b. Notwithstanding the F_{uav} maximization in λ_{uav} plot (it almost reaches $30km$ with $R = 16km$), we observe in α_{uav} plot that the time difference among f_{uav} and $f(g)_{uav}$ solutions is below $1h$.

And third, λ_{total} and α_{total} plots have different outcomes (top and bottom right plots). λ_{total} is always in negative outcomes for every $g \in G_p$. That is because the λ_{total} results only have taken into account the UGV's and UAV's travelling distance. Here, both $f(g)_{ugv}$ and $f(g)_{uav}$ are computed with equal importance from a distance perspective. Nevertheless, if we take the commented motion speed assumption ($V_{ugv} << V_{uav}$), we can see that the UGV's motion speed ($V_{ugv} = 0.13km/h$) is much slower than UAV's motion speed ($V_{uav} = 30km/h$). Then, we can ensure that the UGV will need more time to cover the same distance than the UAV. Consequently, the $f(g)_{ugv}$ optimization will be more significant than the $f(g)_{uav}$ optimization from a time perspective. As we can see in α_{total} plot, the results are always in positive outcomes. Both plots show that the gravitation optimization algorithm computes better solutions (in average) computing the MC gravity point until $R \approx 12$. These statements can be defined as follows:

$$\lambda_{total}(MC) > \lambda_{total}(g) \, \forall g \in \{XC, HC\}, \; R <= 12 \tag{3.20}$$

$$\alpha_{total}(MC) < \alpha_{total}(g) \, \forall g \in \{XC, HC\}, \; R <= 12 \tag{3.21}$$

Therefore, we can assert that R has a key impact on the TERRA performance. In particular, the gravitational optimization algorithm performance will always depend on R and the motion speed of the heterogeneous simple UGV-UAV system. From a distance perspective, the performance is lower as R increases, i. e., λ_{total} increases, because the UGV's travelling distance minimization is less significant than the UAV's travelling distance maximization. Nevertheless, from a time perspective, the performance is higher as R increases, i. e., α_{total} decreases, because the UGV's travelling time minimization is more significant than the UAV's travelling time maximization.

3.4.1.3 *Analysing clustering effects*

The objective is to analyse Lemma 3.4.2 by evaluating the search algorithm presented in subsection 3.2.5, over different clustering settings.

This experiment consists of six parameters configurations in which $\delta = \{2, 4, 8, 16, 32, 64\}$. The rest of the key parameters have been kept constant during the experimentation ($R = 2$ and $N = 64$). Each configuration has been tested over five hundred random problem instances. Figure 3.10 shows the experiment results. From left to right, the plots represent F_{ugv}, F_{uav}, F_{total} and $ChargingStops$ for the three gravity point solutions (XC, HC, MC) and the solution without computing a gravity point. Each plot shows the TERRA solutions in the three gravity point results and the results without a gravity point.

From Figure 3.10, we can highlight three evidences. First, the F_{ugv} plot shows that, the higher δ, the higher are f_{ugv} and $f(g)_{ugv}\ \forall\ g \in G_p$. In fact, the higher δ, the higher is the gravitational optimization algorithm performance because it finds less constraints to compute a junction point near to the gravity point (see subsection 3.2.3). This explains why the $f(g)_{ugv}\ \forall\ g \in G_p$ are slightly lower than f_{ugv} while δ increases. Nevertheless, the higher δ, the higher is the number of charging stops and so, the higher are f_{ugv} and $f(g)_{ugv}\ \forall\ g \in G_p$.

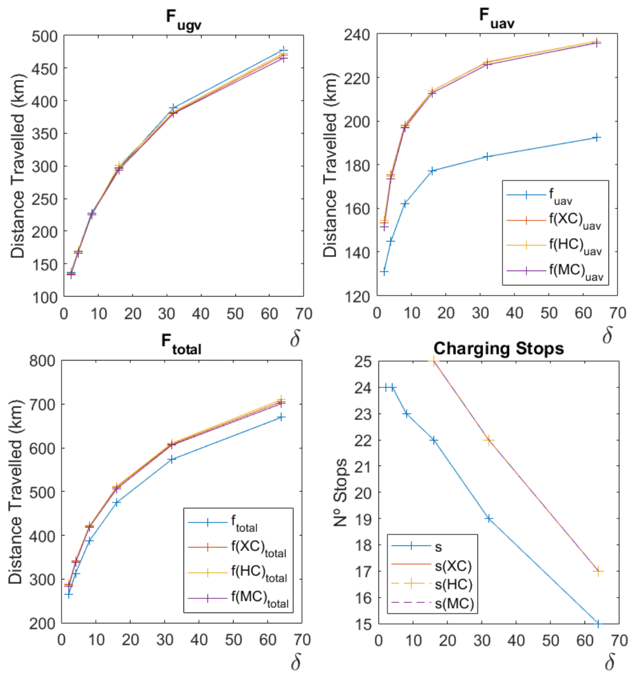


Figure 3.10: Results for the TERRA execution in six different ECU-CSURP configurations with $\delta = \{2, 4, 8, 16, 32, 64\}$, $R = 2$ and $N = 64$. Each configuration has been evaluated over five hundred randomly generated maps. We demonstrate that between two maps with the same key parameters but different δ, TERRA generates a better solution to the map with lower δ.

Second, the F_{uav} plot shows that, the higher δ, the higher are f_{uav} and $f(g)_{uav}\ \forall\ g \in G_p$. This is because of the search algorithm performance, i.e., the higher δ, the lower the probability to schedule a path with intermediate charging stops. Also, we can appreciate two differences among $f(g)_{uav}\ \forall\ g \in G_p$ and f_{uav}. On the one hand, the search algorithm computes the maximum increment of $f(g)_{uav}\ \forall\ g \in G_p$, i.e., $d(p_j, G_p) \forall\ p_j \in P$ in Figure 3.5a. Then, it needs to place the maximum number of charging stops in the UAV sub-tours to accomplish them.

As we can see in the *Charging Stops* plot, the number of charging stops in $f(g)_{uav}\ \forall\ g \in G_p\ (s(XC), s(HC), s(MC))$ is always higher than f_{uav} (s). Also, we can see that this effect decreases as δ increases. On the other hand, if any gravity point is applied, there is no f_{uav} aggravation, so the search algorithm has a greater performance.

The third and last evidence is related to the intermediate charging stops number in the *Charging Stops* plot. Here, we can observe the $\Delta = \delta/R$ relation. We can assert that the search algorithm has a turning point in $\Delta = 2$. In $\Delta \leqslant 2$, the algorithm performance is high because δ is too low compared with the area covered by R. Then, the search algorithm requires a large number of intermediate charging stops to find a UAV sub-tour. In $\Delta \geqslant 2$, the performance decreases because δ is too high compared with R, and so, the algorithm has less probability to find a UAV sub-tour with a minimum set of intermediate charging stops.

Therefore, we can assert that δ has a key impact on the TERRA performance. As F_{total} plot shows, between two problem instances with the same key parameters but different δ, TERRA generates a better solution in the instance with lower δ. This parameter can be very useful to take it into account in the scientific goal's planning task.

3.4.2 *Implications of $\mathbb{R}^3$ environments in TERRA*

We realized two direct consequences in the performance of the algorithm by extending TERRA to $\mathbb{R}^3$. The first is related to the integration of legit and unlegit vertices, because there can be rugged terrains, where the searching of legit vertices can be costly and may reduce the effectiveness in the first and second stages. Therefore, we performed a clustering optimization experiment where we assess the capability of TERRA to find the minimal number of legit vertices required to cover the set of target points. The second is related to the integration of 3Dana [250] into the algorithm. For that, we performed a second experiment to evaluate the overall performance of TERRA.

Both experiments have been performed under the real Mars DTM shown in Figure 3.7. Taking advantage that the 3Dana algorithm aims to find safe routes for the UGV, we wanted to assess the performance of these experiments according to different safety levels in the navigation of the heterogeneous simple UGV-UAV system. These safety levels allows us to describe several risky environment settings through the next two parameters:

- Terrain slope (P): 3Dana uses the terrain slope as a threshold to compute a UGV's path where every point has less slope than this threshold. A high safety level will be determined by a lower slope because the UGV's path will avoid dangerous terrain features.

- Security range (β): represents the distance percentage that is going to be subtracted from a theoretical farthest distance (R_{th}) the UAV can travel without running out of energy, e. g., with $R_{th} = 300$ meters and $\beta = 0.1$ (10%). Then, $R = R_{th} - (R_{th} * \beta) = 270$ meters. The objective of β is to avoid the UAV to run out of energy due to unpredictable environmental conditions.

3.4.2.1 *Clustering optimization experiment*

The first experiment evaluates the first and second stages of TERRA. We have described that the goal of these stages is to minimize the number of legit vertices, denoted as N_{LV}, required to cover the whole set of target points. But, we cannot ensure that N_{LV} is optimal on every distribution. Then, we use the random scenario generator of Appendix A to define the optimal number of legit vertices, known as δ, on every distribution. The random map generator creates every scenario ensuring that N target points are distributed in δ legit vertices, inside a specific radius R and allocated around a real Mars DTM. Here, the δ parameter allows us to define the optimal number of legit vertices to cluster every target point on each scenario, i. e., TERRA can compute, at the very least, δ legit vertices. Therefore, we are able to compare N_{LV} and δ to detect when TERRA has computed an optimized scenario. We have defined $N_{LV} = \delta$ as an optimized scenario.

Table 3.1: Clustering optimization of the first two TERRA's stages in different safety levels over the Mars DTM shown in Figure 3.7. It has been computed a total of 10000 random scenarios for each safety level. SS = scenarios with solution, WS = scenarios without solution, and OS = optimized scenarios. RTime is the runtime in milliseconds.

Level	β (%)	P (°)	#SS	#WS	OS (%)	RTime (ms)
L1	10	20	10000	0	71.0	6.5
L2	30	10	10000	0	62.3	12.2
L3	60	5	10000	0	63.1	16.7
L4	90	1	7241	2759	39.2	26.1

This experiment consists of the execution of ten thousand random scenarios over the Mars DTM with the four different safety levels based on R and P (L1 = low safety, L4 = high safety) displayed in Table 3.1. P is ranged from 20° to 1 ° (plain terrain). β goes from 10% (low security range) to 90 % (high security range). Then, the goal of this experiment is to determine the optimized scenarios percentage in the L1-L4 safety levels. The results in Table 3.1 show that as long as the safety level increases from L1 to L4, the percentage of optimized scenarios (when $N_{LV} = \delta$) decreases. L1 shows the highest percentage of optimized scenarios, because a high P and a low β represents more possibilities to find legit vertices around the target points, and then, to achieve $N_{LV} = \delta$. L4 shows the lowest percentage of optimized scenarios, because of the highest β and the lowest P, so there are hardly any legit vertices around the target points. Also, we can appreciate that the computational time increases as long as it does the safety level, because TERRA finds less legitimate vertices to ensure legit vertices close to the target points. So, if a legit vertex cannot be reached by the UGV, its means that the target point cannot be visited by the UAV and then, the scenario does not have solution. As shows the L4 results, TERRA does not find a solution for 2759 scenarios.

3.4.2.2 *Computational performance Vs. Safety*

The second experiment evaluates the performance of TERRA as a whole, over different safety levels. The performance has been assessed in terms of the computational time taken to solve every scenario and the computed distance travelled by the heterogeneous simple UGV-UAV system.

Table 3.2: Performance of the complete TERRA algorithm in different safety levels over the Mars DTM shown in Figure 3.7. Every level has been tested over the same 100 random scenarios. F_{ugv} = distance travelled by the UGV, F_{uav} = distance travelled by the UAV, SS = scenarios with solution, and WS = scenarios without solution. RTime is the runtime in seconds.

Level	β (%)	P (°)	F_{ugv} (km)	F_{uav} (km)	#SS	#WS	RTime (s)
L1	10	20	10679.2	1096.1	93	7	226.6
L2	30	15	10727.5	1095.7	65	35	485.6
L3	60	10	11807.8	1109.7	4	96	3562.0
L4	90	5	0	0	0	100	0

This experiment consists of the evaluation of one hundred scenarios over the Mars DTM shown in Figure 3.7 with the safety levels displayed in Table 3.2. P is ranged from 20° to 5° (almost a plain terrain). β goes from 10% (low security range) to 90 % (high security range). The results in Table 3.2 shows that as long as the safety level increases from L1 to L4, the travelling distance clearly increases, the number of scenarios with solution decreases and the computational time increases exponentially. The distance travelled by both robotic systems is minimum in the lowest safety level L1 due to the UGV has more ability to avoid obstacles (high P) and the UAV can fly a farther distance (low β). Then, it is easier to reach a legit vertex close to a target point, which in turn minimizes the distance travelled by the robots. Also, the computed scenarios with solution decreases in a higher safety level due to the UGV has less ability to avoid obstacles (low P) and the UAV can fly a shorter distance (high β). Then, it is more difficult to reach a legit vertex. Consequently, the computational time increases because of the UGV's path planning computed by the 3Dana algorithm, i. e., it needs to expand more nodes around the scenario to find a path among the legit vertices.

3.5 SUMMARY

Heterogeneous simple UGV-UAV systems have a brilliant future ahead because of the synergies among UGV and UAV systems. Their composition and functional capabilities makes them a feasible system for exploration applications. However, much effort is required to reach high-level autonomous explorations were both UGV and UAV are fully autonomous systems. In this chapter, we first introduced the formulation of a novel exploration problem (ECU-CSURP) which can be the basis for high-level and complex applications. Then, we described the TERRA algorithm as a method which executes an ordered set of stages to compute a cooperative path planning solution to the ECU-CSURP. This algorithm exploits a particular cooperation synergy of the UGV-UAV system. Furthermore, we provided the extension of the ECU-CSURP and TERRA to $\mathbb{R}^3$ Euclidean spaces. The experimental evaluation of TERRA demonstrates that its performance relies on the battery capacity of the UAVs, the clustering level of the target points in the exploration area, and the complexity of the terrain features.

4 A COOPERATIVE MULTIPLE UGV–UAV(S) TASK PLANNING ALGORITHM FOR THE LAST-MILE DELIVERY PROBLEM

After the research carried out to address the exploration problem through a cooperative simple UGV-UAV path planning algorithm, our interest shifted to investigate the same paradigm in similar problems, and thus extend its scope for future real applications. Throughout this research, we found out that, around 2013, a novel approach emerged to solve the last-mile delivery problem (discussed in subsection 2.2.2) taking advantage of a truck-drone cooperation. This method uses the truck to transport the drones closer to customer locations, where the drones can be launched to perform their 'last-mile delivery'. In fact, we know that logistics companies such as, DHL, Amazon or UPS, are currently working in the development of similar paradigms [205–207]. However, there is still much work to be done to understand which paradigm optimizes the delivery in each scenario.

Here, we realised that the cooperation paradigm used to tackle the ECU-CSURP in chapter 3, had many similarities with this novel approach. Also, we did not find existing algorithms solving the last-mile delivery problem following the TERRA's method. This triggered the research presented in this chapter: a problem formulation as a generalization of the last-mile delivery problem, and the implementation of a cooperative multiple UGV-UAV(s) task planning algorithm to solve it. To assess our algorithm, we have performed an experimental evaluation where we characterized its behaviour in a wide range of problem instances, and we evaluated its performance taking as baseline a state of the art algorithm.

4.1 A GENERALIZATION OF THE LAST-MILE DELIVERY PROBLEM

The last-mile delivery problem that we focus on deploys a heterogeneous multiple UGV-UAVs system, where the UGV carries the UAVs through intermediate points among the customer locations from where the UAVs are able to perform the last-mile delivery. Specifically, the problem that we propose arises from the ECU-CSURP defined in chapter 3, and furthermore, it adds the constraints to become a generalization of the last-mile delivery problem. This problem is formally named as the multiple UAVs and Charging station Vehicle Last-Mile delivery Problem (mUCVLMP), and it is formulated as follows:

(i) An area modelled as an $\mathbb{R}^2$ Euclidean space.

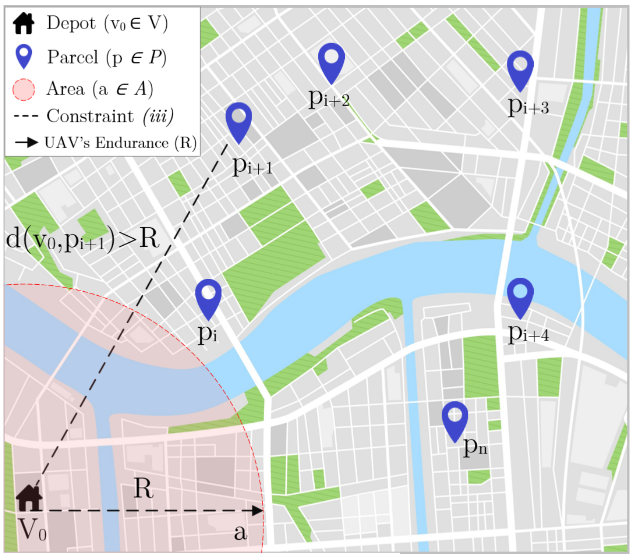


Figure 4.1: A mUCVLMP instance where a set of parcels $p \in P$ and a depot location $v_0 \in V$ are distributed around a $\mathbb{R}^2$ Euclidean space. R denotes the UAV's endurance as the farthest distance the UAV can travel considering a return flight to the take off point. The distance constraint is satisfied with $d(v_0, p_i) > R$.

(ii) A set of parcels which has to be delivered.

(iii) A distance constraint where all the parcels are out of the boundaries of the initial UAV's endurance.

(iv) An heterogeneous multiple UGV-UAVs system modelled as Dubins vehicles [247], which are vehicles with just a single constraint: forward movement at a constant speed.

(v) A UAV's endurance function.

(vi) A depot location where both systems start and finish the mission.

(vii) The last-mile delivery can only be carried out by the UAVs.

(viii) A single UAV can only deliver one parcel simultaneously.

(ix) The UAVs can land on the UGV only if the UGV is already in place.

(x) Only one UAV can take off at a time (parallel landing is not allowed either).

(xi) Only one UAV can land at a time (parallel taking off is not allowed either).

The objective is to plan a set of tasks for the heterogeneous multiple UGV-UAVs system to deliver all the parcels while trying to minimize the total delivery time. Figure 4.1 shows a problem instance with a graphical representation of some of the problem constraints. Constraints (i), (ii) and (iii) are exactly the same as in the ECU-CSURP definition, but defining parcels instead of target points. Constraint (iv) defines that the UGV can carry a team of UAVs instead of only one UAV as in the ECU-CSURP definition. Also, the UAV's movement speeds are constants for the taking off, landing and flying stages. Constraint (v) sets the UAV's endurance function as the farthest distance the UAV can travel with a fully charged battery. To compute the UAV's endurance, we define every parcel delivery as an eight stages procedure, which starts when the UGV arrives to a vertex location from where the UAV has to deliver a parcel:

1. The UAV is on the UGV, so it can charge its battery before to take off (Tc = charging time)
2. If the UAV's battery is full and there is other UAV landing/taking off, it also could wait before to take off (Two = waiting to take off time).
3. It takes off carrying the parcel (To = take off time).
4. It flies to the customer location (Ta = flying time).
5. It delivers the parcel to the customer (Td = delivery time).
6. It flies back to the UGV's location (Tb = flying time).
7. Before landing, it waits until there is a landing windows available (Twl = waiting to land time).
8. It lands on the UGV (Tl = landing time).

Therefore, we denote the delivery time of a parcel as $Ttrip = To + Ta + Td + Tb + Twl + Tl$. If $Ttrip$ is maximum (the UAV drains all of its battery), then $Ta = Tb$ and the UAV's endurance is $R = Ta \cdot V_{uav}$. We assume the UGV does not have energy constraints, so it has enough energy to complete a delivery mission.

The UAV does not drain its energy in Tc and Two

Constraint (vi) denotes that the total delivery time will start and finish at a depot location. Also, we assume that there will not be deliveries from the depot location. Constraint (vii) requires that the parcel has to be delivered by the UAVs. Also, we assume that every UAV can deliver any parcel on board the UGV, so we do not distinguish among parcels. Constraints (viii-xi) set the coordination policies to illustrate a real-world delivery problem. These last constraints represent the relevant differences to the ECU-CSURP definition.

Given the above mUCVLMP formulation, we define the mUCVLMP as a collection of two sub-problems: a first sub-problem where the objective is to find an ordered UGV's path minimizing the UGV's travel time (T_{ugv}), and a second problem focused on minimizing the delays caused by the deliveries (T_{delays}). Therefore, the mUCVLMP objective can be summarized as follows:

$$\text{minime} \;\rightarrow\; T_{total} = T_{ugv} + T_{delays} \tag{4.1}$$

where the objective is to minimize the total delivery time as a function of the UGV's travel time and the time delays experienced in the deliveries.

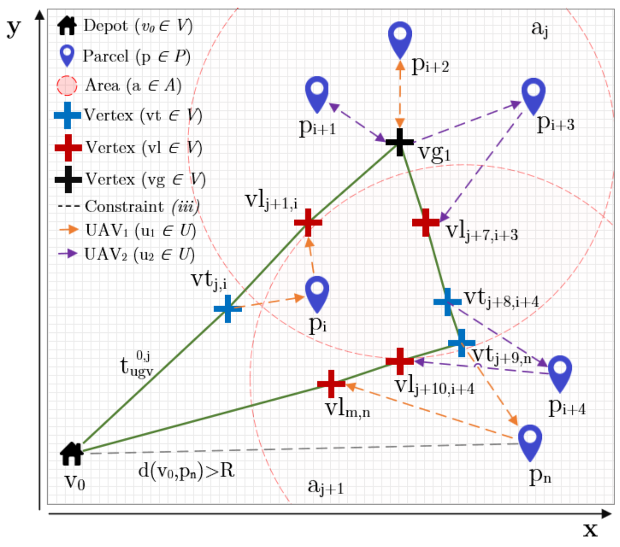


Figure 4.2: A mUCVLMP graphical representation, where a set of parcels $p \in P$, a depot location $v_0 \in V$, a set of areas $a \in A$, and a set of vertices $v \in V$ are placed in a $\mathbb{R}^2$ Euclidean space. The green line is the UGV's path, the orange dashed arrow is the UAV's path of $u_1 \in U$, and the purple dashed arrow is the UAV's path of $u_2 \in U$. Blue crosses are the take off vertices $vt \in V$, red crosses are the landing vertices $vl \in V$, and black crosses are subsets of vertices $vg_z \in V$ in the same location.

Note that: $\forall p_i \in P \; \exists \{vt^k_{j,i}, vl^k_{j,i}\}$ for a UAV u_k.

Note that: $v_j = vt^k_{j,i}$ or $v_j = vl^k_{j,i}$.

Let P denote the set of parcels $\{p_i, \dots, p_n\}$ where a parcel $p_i \in P$ for $i = 1, \dots, n$, and U denote the set of UAVs $\{u_k, \dots, u_s\}$ where a UAV $u_k \in U$ for $k = 1, \dots, s$. Let V denote the set of vertices $V = v_0 \cup Vt \cup Vl$, where a vertex $v_j \in V$ for $j = 0, \dots, m$. Let $v_0 \in V$ represent the depot location[1]. Let Vt denote the set of taking off vertices $\{vt^k_{j,i}, \dots, vt^k_{q,n}\}$ (blue vertices in Figure 4.2) for $i = 1, \dots, n$, $j = 1, \dots, q$ and $k \in \{1, \dots, s\}$. Let Vl denote the set of landing vertices $\{vl^k_{j,i}, \dots, vl^k_{q,n}\}$ (red vertices in Figure 4.2) for $i = 1, \dots, n$, $j = 1, \dots, q$ and $k \in \{1, \dots, s\}$. Let $vt^k_{j,i}$ be the take off vertex $v_j \in V$ of the UAV u_k to deliver a parcel p_i, and $vl^k_{j,i} \in V$ be the landing vertex $v_j \in V$ of the UAV u_k after delivering a parcel p_i. Let Vg_z denote the ordered tuple of vertices which represent the same spatial location. For example, black vertex vg_1 in Figure 4.2 is a set

1 Note that $v_0 \notin Vt \notin Vl$ because there will not be deliveries from the depot.

formed by $\{vt_{j+2,i+1}, vl_{j+3,i+1}, vt_{j+4,i+2}, vl_{j+5,i+2}, vt_{j+6,i+3}\}$[2]. Thus, let V be denoted as $\{v_0, vt^k_{j+1,i}, vl^k_{j+2,i}, \dots, vt^k_{m-1,n}, vl^k_{m,n}\}$ for $i = 1, \dots, n$, $j = 0, \dots, m$ and $k \in \{1, \dots, s\}$. Let the distance constraint be represented as $\{\forall p \in P : d(v_0, p) \leq R\} = \emptyset$ (black dashed line in Figure 4.2). Therefore, let a UGV's path (green line in Figure 4.2) be denoted as an ordered tuple $\{v_0, vt^k_{1,1}, \dots, vl^k_{m,n}, v_0\}$, where it may contain any combination of take off and landing vertices. Let $d_{v_j, v_{j+1}}$ represent the Euclidean distance from v_j to v_{j+1}, and V_{ugv} is the UGV's speed. Then, we define the objective function of the first sub-problem as follows:

$$\text{mininime} \rightarrow T_{ugv} = \sum_{j \in V} t^{j,j+1}_{ugv} \tag{4.2}$$

$$t^{j,j+1}_{ugv} = \frac{d_{j,j+1}}{V_{ugv}} \tag{4.3}$$

The second sub-problem formulates the two type of delays experienced in every delivery: a divorce delay $\Delta Tc^k_{j,i}$ as the UGV's waiting time in $vt^k_{j,i}$ till the UAV u_k finishes its taking off, and the rendezvous delay $\Delta Tw^k_{j,i}$ as the UGV's waiting time in $vl^k_{j,i}$ till the UAV u_k finishes its landing. Therefore, the total delay is formulated as follows:

$$\text{minimize} \rightarrow T_{delays} = \sum_{j=1}^{m} \sum_{i=1}^{n} \sum_{k=1}^{s} \Delta Tc^k_{j,i} \cdot X_j + \Delta Tw^k_{j,i} \cdot \overline{X_j} \tag{4.4}$$

where X_j is a binary variable which takes a value 1 if $v_j \in Vt$, or if it is the last take off vertex of an ordered tuple vg_z; or it takes a value 0 if $v_j \in Vl$, or it is the last landing vertex of vg_z.

Let $Tc^k_{j,i}$ represent the required charging time of the UAV u_k in the vertex $vt^k_{j,i}$, before the taking off, indispensable to have enough energy to deliver the parcel p_i. Let $Two^k_{j,i}$ be the waiting time of a UAV u_k with a full battery only when there is another UAV taking off at the same time, and To^k be the constant taking off time of a UAV u_k. Then, we define the divorce delay as follows:

$$\Delta Tc^k_{j,i} = Tc^k_{j,i} + Two^k_{j,i} + To^k \tag{4.5}$$

To compute $Tc^k_{j,i}$ we measure the drained energy of a UAV in time terms. We assume an ideal scenario where the UAV's charging ratio is equal to the UAV's discharging ratio, so the time spent charging the battery is equal to the time spent discharging it. This assumption aims to avoid the integration of physical constraints, such as wind speed or

2 Note that Vg_z notation is described to facilitate the representation of vertices with the same location.

atmospheric density, which may lead to a very complex UAV's energy function. For instance, given an ordered tuple $\{vt^k_{j,i}, vl^k_{j+1,i}, vt^k_{j+2,i+1}\}$, we have that $Tc^k_{j+2,i+1} = Ttrip^k_{j+1,i}$. We assume that every UAV starts with a fully charged battery, so the first charging time of each UAV is $Tc^k_{i,j} = 0$. Also, we assume that every UAV can recharge its battery while waiting on the UGV for its next delivery. Therefore, the charging time in a vertex is defined by the past actions of the UAV. We define two actions: the UAV is discharging energy in a delivery and the UAV is charging on the UGV. Then, the charging time is formulated as follows:

$$Tc^k_{j,i} = \sum_{h=1}^{j-1} \left(\left(\Delta Tc^{U(h)}_{h,P(h)} \cdot X_h + \Delta Tw^{U(h)}_{h,P(h)} \cdot \overline{X_h} \right) \cdot Y^{U(h)}_h + t^{h,h+1}_{ugv} \cdot Y^{U(h)}_{h+1} \right) \tag{4.6}$$

$$0 \leq Tc^k_{j,i} \leq 2(R \cdot V^k_{uav}) + To^k + Td^k + Tl^k \tag{a}$$

$$X_h \in \{0,1\}, \ \forall h \in \{1,\ldots,j-1\} \tag{b}$$

$$X_h = 1 \leftarrow v_h \neq v_{h+1} \vee v_h \exists Vt \tag{c}$$

$$X_h = 0 \leftarrow v_h \neq v_{h+1} \vee v_h \exists Vl \tag{d}$$

$$Y^{U(h)}_h \in \{1,-1\}, \ \forall h \in \{1,\ldots,j-1\} \tag{e}$$

$$Y^{U(h)}_h = 1 \leftarrow U(h) = k \tag{f}$$

$$Y^{U(h)}_h = -1 \leftarrow U(h) \neq k \tag{g}$$

where $U : U(h) \rightarrow u_k$ is the mapping function which returns the corresponding UAV u_k to the vertex v_h, R is the UAV's endurance, and $P : P(h) \rightarrow p_i$ is a mapping function which returns the corresponding parcel p_i to the vertex v_h. The inequality 4.6a sets that the charging time is always positive and less than the UAV's endurance. Also, X_h (4.6b) is a binary variable which takes a value 1 (4.6c) if v_h is in Vt or it is the last vertex of an ordered tuple vg_z; or it takes a value 0 (4.6d) if v_h is in Vl or it is the last vertex of vg_z. Last, Y_h (4.6e) is a binary variable which takes value 1 (4.6f) if the UAV u_h deployed in v_h is equal to the UAV u_k deployed in $v^k_{j,i}$ (action: the UAV is discharging energy) or it takes value -1 (4.6g) if the deployed UAV u_h is different to u_k (action: the UAV is charging energy).

Once the UAV u_k has taken off at $vt^k_{j,i}$ to deliver the parcel p_i, the UGV travels to the rendezvous vertex $vl^k_{j,i}$, where it waits till the UAV finishes the delivery and lands on the UGV, i. e., a rendezvous delay. There are two types of rendezvous delays: simple and complex rendezvous.

A simple rendezvous is represented by any UGV's path as an ordered tuple $\{vt^k_{j,i}, vl^k_{j+1,i}\}$ (see $\{vt^k_{j,i}, vl^k_{j+1,i}\}$ in Figure 4.2), where there are not intermediate vertices, and it is computed as follows:

$$Tw^k_{j,i} = t^{j,i,k}_{uav} - t^{H(vl^k_{j,i}),j}_{ugv} \tag{4.7}$$

$$x < y \; \forall \{vt^k_{x,i}, vl^k_{y,i}\} \in V \tag{a}$$

$$Tw^k_{j,i} \geq \; Tl^k, \;\; \forall \, vl^k_{j,i} \in V \tag{b}$$

where t_{uav} is the UAV's travels time, t_{ugv} is the UGV's travel time and $H : Vl \rightarrow Vt \; \forall i \in \{1, \ldots, n\}$ is the mapping function to obtain the corresponding index of the take off vertex of a particular landing vertex, i. e., $H(vl^k_{y,i}) = x$. The inequality 4.7a ensures that the UGV visits $vt^k_{x,i}$ before its rendezvous $vl^k_{y,i}$ for every parcel delivery, and the inequality 4.7b sets that the UGV will always arrive to $vl^k_{j,i}$ before the UAV begins the landing stage (Constraint ix). This ensures a reliable coordination of the heterogeneous multiple UGV-UAVs system.

Let A denote the set of areas $\{a_1, \ldots, a_m\}$, where an area $a_j \in A$ represents the set of covered parcels $\{p_1, \ldots, p_n\}$. For instance, $a_j = \{p_i, p_{i+1}, p_{i+2}, p_{i+3}\}$ in Figure 4.2. Also, let a UAV's path be denoted as an ordered tuple $\{vt^k_{j,i}, p_i, vl^k_{j+1,i}\}$ (see orange and purple dashed arrows in Figure 4.2). Let $d_{vt^k_{j,i},p_i}$ represent the Euclidean distance from $vt^k_{j,i}$ to p_i, and V^k_{uav} the speed of the UAV u_k. Let Td^k be the constant delivery time of a UAV u_k. Let $Twl^k_{j,i}$ be the waiting time of a UAV u_k to land in $vl^k_{j,i}$, if and only if there is another UAV already landing. Let Tl^k be the constant landing time of a UAV u_k. Then, we define a UAV's travel time t_{uav} as follows:

$$t^{j,i,k}_{uav} = g^k_{H(vl^k_{j,i}),p_i} + Td^k + g^k_{vl^k_{j,i},p_i} + Twl^k_{j,i} + Tl^k \tag{4.8}$$

$$g^k_{v_j,p} = \frac{d_{v_j,p}}{V^k_{uav}} \tag{4.9}$$

Note that t_{uav} does not integrate To^k, because it is part of the divorce delay described in Equation 4.5, which means that the UGV will start moving from v_j to v_{j+1} as soon as the UAV has began to take off at v_j.

A complex rendezvous is represented by any other UGV's path which has intermediate vertices in an ordered tuple $\{vt^k_{j,i}, \ldots, vl^k_{j+1,i}\}$

(see vertices of p_{i+4} and p_n in Figure 4.2). In this way, a total rendezvous delay $\Delta Tw^k_{j,i}$ is defined for any type of rendezvous as follows:

$$\Delta Tw^k_{j,i} = \; Tw^k_{j,i} + \Delta Tc^k_{j,i} \cdot Z_j - \left(\sum_{\sigma}^{j-1} \Delta Tc^{U(h)}_{h,P(h)} \cdot X_h + \Delta Tw^{U(h)}_{h,P(h)} \cdot \overline{X_h} \right) \cdot \overline{Z_j} \tag{4.10}$$

$$\sigma \leftarrow h = H(vl^k_{j,i}) \qquad \text{(a)}$$

$$\Delta Tw^k_{j,i} \geq \; Tl^k, \;\; \forall \, vl^k_{j,i} \in V \qquad \text{(b)}$$

$$Z_j \in \; \{0,1\}, \;\; \forall \, j \in \{1, \dots, m\} \qquad \text{(c)}$$

$$Z_j = \; 1 \leftarrow H(vl^k_{j,i}) = j \qquad \text{(d)}$$

$$Z_j = \; 0 \leftarrow H(vl^k_{j,i}) \neq j \qquad \text{(e)}$$

where the Equation 4.10a is used to simplify the summation index of Equation 4.10, the inequality 4.10b sets that the UGV will always arrive to $vl^k_{j,i}$ before the UAV begins the landing stage (Constraint ix), Z_j (4.10c) is a binary variable which takes value 1 (4.10d) if the take off index $H(vl^k_{j,i})$ is equal to the landing index of $vl^k_{j,i}$ and 0 (4.10e) otherwise, and the X_j binary variable and functions $U(h)$ and $P(h)$ are the same as described in Equation 4.6.

4.2 THE COURIER ALGORITHM FOR $\mathbb{R}^2$ EUCLIDEAN SPACES

The implemented task planning algorithm is formally named COoperative Unmanned deliveRIEs planning algoRithm (COURIER). This algorithm implements a UGV-UAV cooperation paradigm with a heterogeneous multiple UGV-UAVs system to solve the mUCVLMP. Here, the UGV is a moving charging station which carries the UAVs along with parcels around locations, from where the UAVs can deliver the parcels to the customers. Each UAV, with enough energy, is a qualified candidate to perform the delivery. A delivery contains two vertices: the take off and the landing vertex. In this way, the UAV starts the delivery in the take off vertex, where it separates from the UGV. Then, the UAV finishes the delivery in the landing vertex, where it rendezvous with the UGV. Each UAV has its own charging station on board the UGV, so it can recharge the battery at any time, as long as it is docked on the UGV. The mUCVLMP finishes when every parcel has been delivered, and the heterogeneous multiple UGV-UAVs system reaches the depot location.

This algorithm replicates the first four stages implemented by TERRA as shows Figure 4.3, but it builds a new fifth stage for computing the complete task planning of the heterogeneous multiple UGV-UAVs system. At the fourth stage, TERRA returned the UGV's directed path, where every vertex $v_j \in V$ has its own area $a_j \in A$,

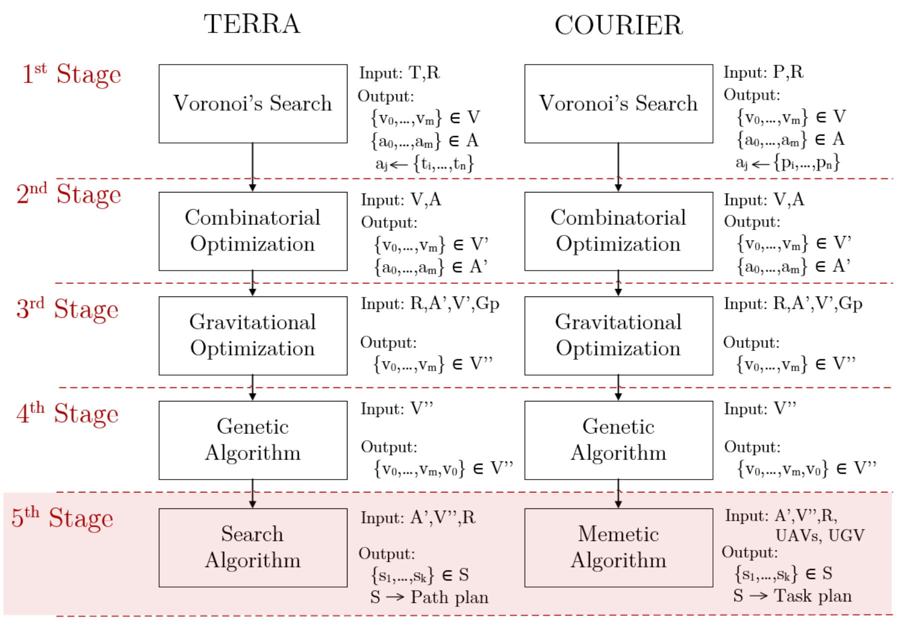

TERRA
COURIER
1st Stage
2nd Stage
3rd Stage
4th Stage
5th Stage
Voronoi's Search
Input: T,R
Output: {v0,...,vm} ∈ V {a0,...,am} ∈ A aj ← {ti,...,tn}
Input: P,R
Output: {v0,...,vm} ∈ V {a0,...,am} ∈ A aj ← {pi,...,pn}
Combinatorial Optimization
Input: V,A
Output: {v0,...,vm} ∈ V' {a0,...,am} ∈ A'
Gravitational Optimization
Input: R,A',V',Gp
Output: {v0,...,vm} ∈ V''
Genetic Algorithm
Input: V''
Output: {v0,...,vm,v0} ∈ V''
Search Algorithm
Input: A',V'',R
Output: {s1,...,sk} ∈ S S → Path plan
Memetic Algorithm
Input: A',V'',R, UAVs, UGV
Output: {s1,...,sk} ∈ S S → Task plan

Figure 4.3: Comparison between the five stages implemented by TERRA and COURIER. Both algorithms implement the same first four stages. However, the fifth stage of TERRA (Search Algorithm) returns a path planning for the heterogeneous simple UGV-UAV system, whereas the fifth stage of COURIER (Memetic Algorithm) returns a task planning for the heterogeneous multiple UGV-UAVs system.

and every area is a set of the target points $a_j \leftarrow \{t_i, \ldots, t_n\}$ covered by this vertex. In the same way, COURIER achieves its fifth stage with the computed UGV's directed path and the set of areas covering the parcels $a_j \leftarrow \{p_i, \ldots, p_n\}$. Then, it splits up the problem into sub-tours, where each sub-tour is computed separately. A sub-tour $s_j \in S$ is represented by its vertex v_j, the two adjacent vertices $\{v_{j-1}, v_{j+1}\}$ and the set of parcels covered by its area $a_j \leftarrow \{p_x, \ldots, p_y\}$. At this point, subsection 4.2.1 presents a geometrical rendezvous method to describe and compute every rendezvous between the UGV and the UAVs.

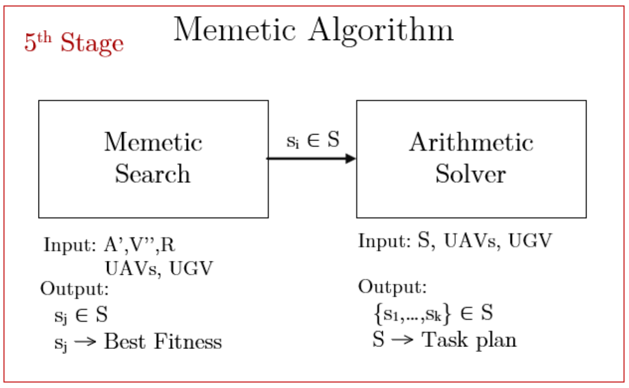

5th Stage
Memetic Algorithm
Memetic Search
si ∈ S
Arithmetic Solver
Input: A',V'',R UAVs, UGV
Output: sj ∈ S sj → Best Fitness
Input: S, UAVs, UGV
Output: {s1,...,sk} ∈ S S → Task plan

Figure 4.4: The fifth stage implemented by COURIER. It is based on a memetic search to look for an approximate solution and an arithmetic solver to compute the final task planning of the solution.

Then, COURIER builds a memetic algorithm based on a memetic search and an arithmetic solver as Figure 4.4 shows. Thus, subsection 4.2.2 describes the memetic search of the fifth stage to look for the plan that minimizes the objective function of mUCVLMP (Equation 4.1). The memetic search implements a fitness function that computes approximated solutions to the problem. Thereafter, subsection 4.2.3 presents the implemented solver to compute the complete task planning from that approximated solution.

4.2.1 *The geometrical rendezvous method*

The geometrical rendezvous method provides a robust geometrical formulation to compute the task planning of any rendezvous at the fifth stage. For that, we need a procedure to compute any divorce (ΔTc) and rendezvous delay (ΔTw), and thus, to obtain the total delay T_{delay} (Equation 4.4) and the total time T_{total} (Equation 4.4) of any sub-tour in the mUCVLMP. Once the algorithm knows these times, it can build the task plan of a sub-tour. In the following, we describe the geometrical rendezvous method to get the task plan of a simple sub-tour. Then, we explain in detail the formulation to solve any sub-tour.

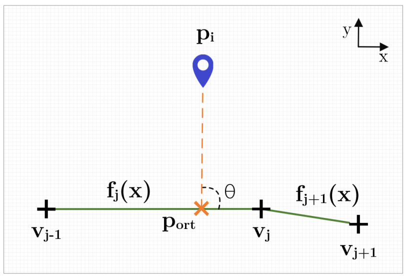
p_{ort} is the closest point of the parcel to the UGV's path, and so, the minimum UAV's travel time to deliver a parcel

This method represents each parcel as an object with three geometrical features (see Figure 4.5a): the Euclidean lines $f_j(x)$ and $f_{j+1}(x)$ of the UGV's directed path of the sub-tour, its orthogonal point p_{ort} in the UGV's directed path, and the angle θ formed by the orthogonal point with the x-axis. These three features enable the computation of the geometrical rendezvous of a parcel, which is characterized by the coordinates of the vertices $vt^k_{j,i}$ and $vl^k_{j+x,i}$ resulting from the following system of equations:

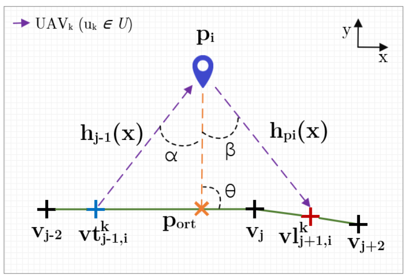
$$\left.\begin{aligned} f_j(x)\ \forall j \in s_j \\ h_i(x) = tan(\theta + \phi) \cdot (x - x_{pi}) + y_{pi} \end{aligned}\right\} \tag{4.11}$$

where $f_j(x)$ represents any line of the UGV's directed path in sub-tour s_i, $h_i(x)$ is the UAV's directed path, ϕ is an offset angle which takes value α to compute the take off vertex $vt^k_{j,i}$ or β to compute the land vertex $vl^k_{j,i}$ (see Figure 4.5b), and the tuple (x_{pi}, y_{pi}) with the Cartesian coordinates of the parcel.

Given the take off vertex $vt^k_{j,i}$ and the landing vertex $vl^k_{j+x,i}$ of a rendezvous, the geometrical method can compute the divorce delay and the rendezvous delay as in the mUCVLMP (Equation 4.5 and Equation 4.7). For instance, Equation 4.12 is an example of computing the divorce delay $\Delta Tc^k_{j-1,i}$ for the vertex $vt^k_{j-1,i}$ in Figure 4.5b, and



(a) The key features of a parcel: $f_j(x)$, $f_{j+1}(x)$, p_{ort} and θ.



(b) An example of a rendezvous.

Figure 4.5: An example of the features of a parcel in the geometrical rendezvous method. The α and β angles are used by the rendezvous method to compute the tangent of the lines $h_{j-1}(x)$ and $h_{pi}(x)$.

Equation 4.13 is an example for the rendezvous delay $\Delta Tw^k_{j+1,i}$ for the vertex $vl^k_{j+1,i}$.

$$\begin{aligned} \Delta Tc^k_{j-1,i} &= Tc^k_{j-1,i} + Two^k_{j-1,i} + To^k \\ Tc^k_{j-1,i} &= Tc^k_{j-2,i} - t^{j-2,j-1}_{ugv} \end{aligned} \tag{4.12}$$

$$\begin{aligned} \Delta Tw^k_{j+1,i} &= t^{j+1,i,k}_{uav} - t^{vt^k_{j-1,i}, vl^k_{j+1,i}}_{ugv} \\ t^{j+1,i,k}_{uav} &= g^k_{vt^k_{j-1,i}, p_i} + Td^k + g^k_{vl^k_{j+1,i}, p_i} + Twl^k_{j+1,i} + Tl^k \\ t^{j-1,j+1}_{ugv} &= \frac{d_{j-1,j} + d_{j,j+1}}{V_{ugv}} \end{aligned} \tag{4.13}$$

where $Two^k_{j-1,i} = 0$ because there is no other UAV taking off at the same vertex, and $Twl^k_{j+1,i} = 0$ because there is no other UAV landing at the same vertex.

Once obtained the divorce and rendezvous delays, we can compute the total delay (T_{delay}) and the total time (T_{total}) of the sub-tour as follows:

$$T_{total} = T_{ugv} + T_{delays} \tag{4.14}$$

$$T_{ugv} = t^{v_{j-2}, vt^k_{j-1,i}}_{ugv} + t^{vt^k_{j-1,i}, v_j}_{ugv} + t^{v_j, vl^k_{j+1,i}}_{ugv} + t^{v_j, vl_{j+2}}_{ugv} \tag{4.15}$$

$$T_{delays} = \Delta Tc^k_{j-1,i} + \Delta Tw^k_{j+1,i} \tag{4.16}$$

At this point, the algorithm can build the total task plan of the sub-tour. Figure 4.6 shows the task plan to deliver the parcel p_i in Figure 4.5b. At start, the UGV moves from vertex v_{j-2} to $vt^k_{j-1,i}$ ($t^{j-2,j-1}_{ugv}$), then it waits till the UAV$_1$ takes off ($\Delta Tc^k_{j-1,i}$). At the vertex $vt^k_{j-1,i}$, the UAV$_1$ charges the battery required ($Tc^1_{j-1,i}$) to perform the delivery, then it takes off (To^1). After the taking off, the UGV goes to vertex

$vl^k_{j+1,i}$ passing through v_j ($t^{j-1,j+1}_{ugv}$). Then, it waits till the UAV$_1$ has landed ($\Delta Tw^k_{j+1,i}$). While the UGV is moving, the UAV$_1$ goes to the parcel location (g^1_{j-1,p_i}) , performs the delivery (Td^1), returns to the UGV's location ($g^1_{p_i,j+2}$) and lands on the UGV (Tl^1). At the end, the UGV moves to v_{j+2} to continue with the deliveries.

Note that g^1_{j-1,p_i} is the simplified form of $g^1_{vt^1_{j-1,i},p_i}$

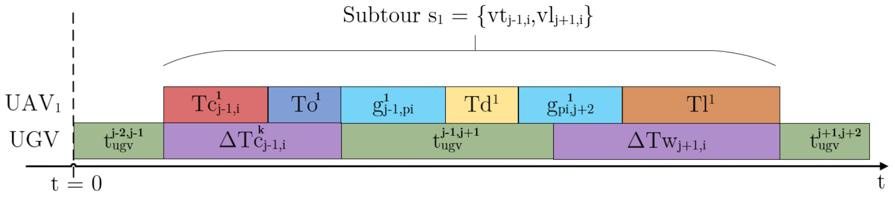
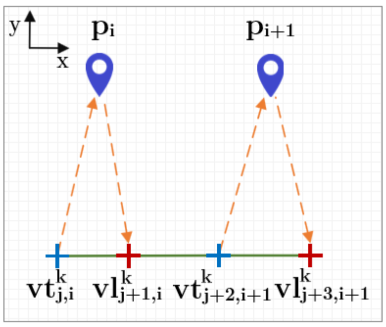


Figure 4.6: A task plan example of a simple rendezvous with one UGV and one UAV. It represents the divorce delay ΔTc and the rendezvous delay ΔTw used to compute the total rendezvous time of Figure 4.5b.

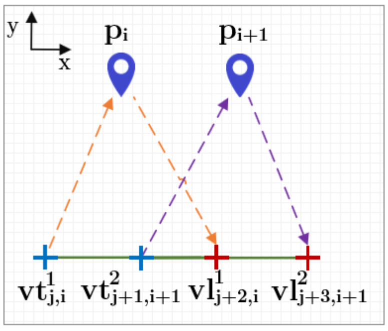
So far, we have described how this method works to compute the task plan of a simple sub-tour. Nevertheless, there might exist sub-tours with any possible combination of vertices. In this way, this method defines a robust model to represent all the vertices combinations that may occur in mUCVLMP. These rendezvous combinations are also called as vertices entanglements. In order to describe the whole spectre of combinations, a scalability approach is defined, where complex entanglements can be computed as a conjunction of elemental entanglements. These elemental entanglements represent all the basic combinations that may experience two parcels in a sub-tour, and are split up into two classes: distance entanglements and time entanglements. In the following, we describe the formulation of both classes to compute the related delays, and so, the total rendezvous delay (T_{delays}) and total time (T_{total}) of the sub-tour. In this way, T_{ugv} is always computed following Equation 4.15. Also, we assume that the first vertex of the sub-tour is always $vt^k_{j,i}$, and all UAVs start with a full battery i. e., $Tc^k_{j,i} = 0$.

4.2.1.1 *Distance entanglements*

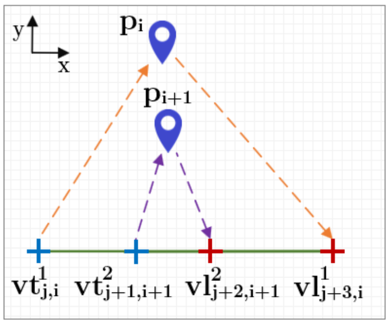
The distance entanglements are those whose vertices can be ordered by the Euclidean distance, thus, every vertex $v_q \neq v_p \forall v_j \in s_z$, where $s_z \in S$ is the ordered tuple of vertices linked to the sub-tour $a_z \in A$. There are three distance entanglements, as Figure 4.7 shows: (a) **No distance entanglement**, (b) **Semi distance entanglement** and (c) **Total distance entanglement**. All of them have in common that $Two^k_{j,i} = 0$, because there is not other UAV taking off at the same vertex, and all $Twl^k_{j,i} = 0$, because there is not another UAV landing at the same vertex.



(a) No distance entanglement



(b) Semi distance entanglement



(c) Total distance entanglement

Figure 4.7: The distance entanglements of the geometrical rendezvous method.

The **No distance entanglement** (Figure 4.7a) is the most elemental one, because it does not exist entanglement. Both parcels p_i and p_{i+1} can be delivered by any UAV $u_k \in U$, and the rendezvous delay can be computed as in Equation 4.13. Instead, the divorce delay will vary depending on the selected UAVs to deliver the parcels. Then, there are two scenarios, where $u_1 = u_2$ or $u_1 \neq u_2$.

The scenario where $u_1 = u_2$ computes T_{delays} as follows:

$$
\begin{aligned}
T_{delays} &= \Delta Tc^k_{j,i} + \Delta Tw^k_{j+1,i} + \Delta Tc^k_{j+2,i+1} + \Delta Tw^k_{j+3,i} \qquad (4.17)\\
\Delta Tc^k_{j,i} &= Tc^k_{j,i} + Two^k_{j,i} + To^k\\
\Delta Tw^k_{j+1,i} &= Tw^k_{j+1,i} = t_{uav}^{j+1,i,k} - t_{ugv}^{vt^k_{j,i},vl^k_{j+1,i}}\\
\Delta Tc^k_{j+2,i+1} &= Tc^k_{j+2,i+1} + Two^k_{j+2,i+1} + To^k\\
Tc^k_{j+2,i+1} &= To^k + t_{ugv}^{vt^k_{j,i},vl^k_{j+1,i}} + Tw^k_{j+1,i} + t_{ugv}^{vl^k_{j+1,i},vt^k_{j+2,i+1}}\\
\Delta Tw^k_{j+3,i} &= Tw^k_{j+3,i} = t_{uav}^{j+3,i,k} - t_{ugv}^{vt^k_{j+2,i+1},vl^k_{j+3,i+1}}
\end{aligned}
$$

The scenario where $u_1 \neq u_2$ computes T_{delays} as follows:

$$\begin{aligned}
T_{delays} &= \Delta Tc^1_{j,i} + \Delta Tw^1_{j+1,i} + \Delta Tc^2_{j+2,i+1} + \Delta Tw^2_{j+3,i} \qquad (4.18)\\
\Delta Tc^1_{j,i} &= Tc^1_{j,i} + Two^1_{j,i} + To^1\\
\Delta Tw^1_{j+1,i} &= Tw^1_{j+1,i} = t_{uav}^{j+1,i,1} - t_{ugv}^{vt^1_{j,i},vl^1_{j+1,i}}\\
\Delta Tc^2_{j+2,i+1} &= Tc^2_{j+2,i+1} + Two^2_{j+2,i+1} + To^2\\
Tc^2_{j+2,i+1} &= -\Delta Tc^1_{j,i} - t_{ugv}^{vt^1_{j,i},vl^1_{j+1,i}} - \Delta Tw^1_{j+1,i} - t_{ugv}^{vl^1_{j+1,i},vt^2_{j+2,i+1}}\\
\Delta Tw^2_{j+3,i} &= Tw^2_{j+3,i} = t_{uav}^{j+3,i,2} - t_{ugv}^{vt^2_{j+2,i+1},vl^2_{j+3,i+1}}
\end{aligned}$$

The **Semi distance entanglement** (Figure 4.7b) describes an interlacing between $vl_{j+2,i}$ of the first parcel and $vt_{j+1,i+1}$ of the second parcel. Thus, the UAVs delivering the parcels must be different. So, there is only one possible scenario, where $u_1 \neq u_2$:

$$\begin{aligned}
T_{delays} &= \Delta Tc^1_{j,i} + \Delta Tc^2_{j+1,i+1} + \Delta Tw^1_{j+2,i} + \Delta Tw^2_{j+3,i+1} \qquad (4.19)\\
\Delta Tc^1_{j,i} &= Tc^1_{j,i} + Two^1_{j,i} + To^1\\
\Delta Tc^2_{j+1,i+1} &= Tc^2_{j+1,i+1} + Two^2_{j+2,i+1} + To^2\\
Tc^2_{j+1,i+1} &= -\Delta Tc^1_{j,i} - t_{ugv}^{vt^1_{j,i},vt^2_{j+1,i+1}}\\
\Delta Tw^1_{j+2,i} &= Tw^1_{j+1,i} - \Delta Tc^2_{j+1,i+1}\\
\Delta Tw^2_{j+3,i+1} &= Tw^2_{j+3,i+1} - \Delta Tw^1_{j+2,i}
\end{aligned}$$

The **Total distance entanglement** (Figure 4.7c) describes an interlacing where the second parcel is delivered during the delivery of the first parcel. Also, the UAVs delivering the parcels must be different, and so, there is only one possible scenario, where $u_1 \neq u_2$:

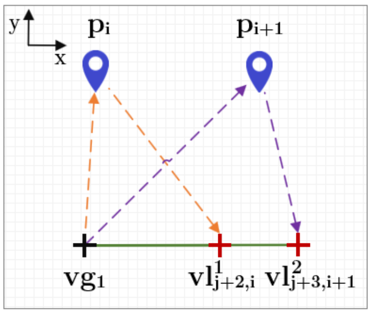
$$\begin{aligned}
T_{delays} &= \Delta Tc^1_{j,i} + \Delta Tc^2_{j+1,i+1} + \Delta Tw^2_{j+2,i+1} + \Delta Tw^1_{j+3,i} \qquad (4.20)\\
\Delta Tc^1_{j,i} &= Tc^1_{j,i} + Two^1_{j,i} + To^1\\
\Delta Tc^2_{j+1,i+1} &= Tc^2_{j+1,i+1} + Two^2_{j+2,i+1} + To^2\\
Tc^2_{j+1,i+1} &= -\Delta Tc^1_{j,i} - t_{ugv}^{vt^1_{j,i},vt^2_{j+1,i+1}}\\
\Delta Tw^2_{j+2,i+1} &= Tw^2_{j+2,i+1} = t_{uav}^{j+2,i+1,2} - t_{ugv}^{vt^2_{j+1,i+1},vl^2_{j+2,i+1}}\\
\Delta Tw^1_{j+3,i} &= Tw^1_{j+3,i} - \Delta Tc^2_{j+1,i+1} - \Delta Tw^2_{j+2,i+1}
\end{aligned}$$

4.2.1.2 *Time entanglements*

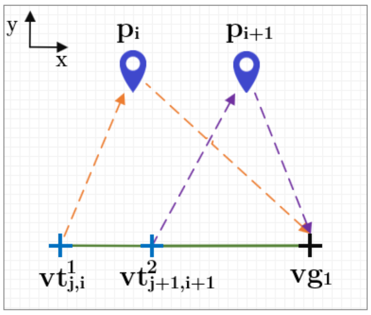
The time entanglements are characterized by having some vertices with the same Cartesian coordinates (see vg_1 in Figure 4.8a-d), so they can not be ordered in s_z by the Euclidean distance. Instead, they need to be ordered in time terms. The group of vertices sharing the same spatial location is denoted as $vg_z \in Vg$, where Vg represents the set of groups of vertices in a sub-tour. Then, as well as the divorce and rendezvous delays, these sub-tours have to compute the total delay of the group Twg_z.

A sub-tour can have several groups vg_z

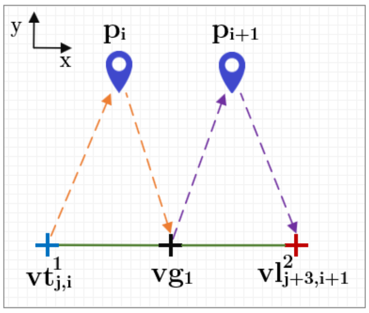
The concept of Twg_z is the same as ΔTc and ΔTw, but for a group of vertices



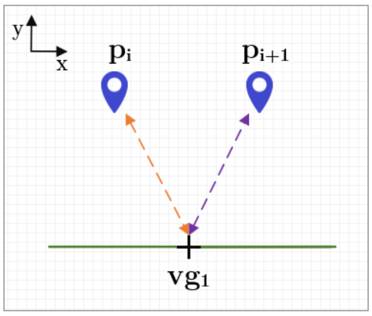
(a) Take off time entanglement



(b) Land time entanglement



(c) Semi time entanglement



(d) Total time entanglement

Figure 4.8: The time entanglements of the geometrical rendezvous method, where all of them have a group of vertices $vg_1 \in Vg$ in the sorted sub-tour s_1.

In this way, each time entanglement presents a vg_1 with a different problematic, and so, this method describes a different solution for each one. For that, it defines $Two^k_{j,i}$ as an additional waiting time, where a UAV has to wait, even with a full battery, to take off. Also, it introduces $Twl^k_{j,i}$ as another waiting time, where a UAV has to loiter before landing on the UGV. Therefore, any complex time entanglement can be approached as a conjunction of multiple elemental types. As Figure 4.8 shows, there are four elemental types: (a) **Take off time entanglement**, (b) **Land time entanglement**, (c) **Semi time entanglement** and (d) **Total time entanglement**.

The **Take off time entanglement** (Figure 4.8a) arises the problem of selecting the take off order of the UAVs involved in the group of vertices $vg_1 = \{vt^k_{j,i}, vt^k_{j+1,i+1}\}$. In the following we explain the process to solve vg_1 and the whole sub-tour. To clarify this explanation, we will use Figure 4.9 as an example of a task plan for vg_1 in Figure 4.8a. There, we can observe that UAV$_1$ takes off before UAV$_2$, and so, UAV$_2$ has to wait ($Two^2_{j+1,i+1}$).

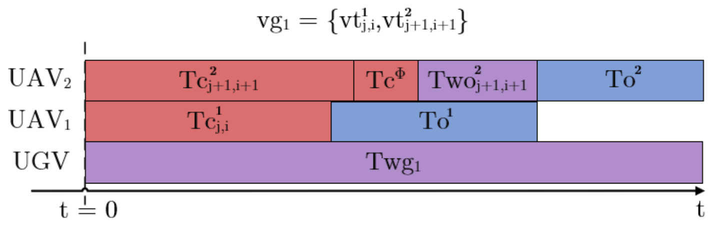


Figure 4.9: A task plan example, with one UGV and two UAVs, for computing vg_1 in the Take off entanglement of Figure 4.8a. The UAV$_2$ has to wait for the take off with $Two^2_{j+1,i+1} > 0$. Also, note that the total group delay $Twg_1 = Tc^2_{j+1,i+1} + Tc^{\Phi} + Two^2_{j+1,i+1} + To^2$.

First, let C^k_j denote the function used to sort the vertices in vg_1, and s_1 represent the ordered tuple of vertices. Then, the criterion set to sort the vertices is shown in Equation 4.21. For instance, Figure 4.9 shows that $vg_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}\}$.

$$\begin{aligned} C^k_j &= To^k + t^{j,k}_{uav} - Tc^k_j - Two^k_j, \qquad \forall\, j \in vg_1 \\ vg_1 &= \{z \in \{1, \dots, m-1\} : C^k_z > C^k_{z+1})\} \\ s_1 &= \{vg_1, vl^1_{j+2,i}, vl^2_{j+3,i+1}\} \end{aligned} \tag{4.21}$$

Constraint (x): only one UAV can take off at a time

Second, this method formulates $Two^k_j \;\forall\; j > 1 \in s_1$ to fulfil Constraint (x) of mUCVLMP as in Equation 4.22. For instance, Figure 4.9 shows that the UAV$_2$ has to wait to take off, so $Two^2_{j+1,i+1} > 0$.

$$\begin{aligned} Q(j,k) &= Tc^k_j - (Tc^k_{j-1} + Two^k_{j-1} + To^k) \\ Two^k_j &= \begin{cases} |Q(j,k)| & \text{, if } Q(j,k) < 0 \\ 0 & \text{, otherwise} \end{cases} \end{aligned} \tag{4.22}$$

Third, if the UAV's battery level is not full after Tc^k_j, and $Two^k_j > 0$, this method introduces the following rule to transform the Two^k_j time into additional charging time Tc^{Φ} as in Equation 4.23. The idea is to keep charging instead of waiting and doing nothing. Figure 4.9 shows Tc^{Φ} as the additional charging time to fill up the battery.

$$\begin{aligned} Tc^{\Phi} &= \begin{cases} Two^k_j & \text{, if } R^k > Two^k_j \\ R^k & \text{, otherwise} \end{cases} \\ Tc^k_j &= Tc^k_j + Tc^{\Phi} \\ Two^k_j &= Two^k_j - Tc^{\Phi} \end{aligned} \tag{4.23}$$

where R^k denotes the battery left after the charging time Tc^k_j.

Four and last, the ordered tuple $s_1 = \{vg_1, vl^1_{j+2,i}, vl^2_{j+3,i+1}\}$ and the computed Two^k_j and Tc^k_j in the previous steps, enable the method to compute the group delay Twg_1. Thus, we can obtain the total rendezvous delay T_{delays} for the Take off time entanglement. For instance, $Twg_1 = Tc^2_{j+1,i+1} + Tc^{\phi} + Two^2_{j+1,i+1} + To^2$ in Figure 4.9. The following Equation 4.24 shows the formulation to compute T_{delays} in a Take off time entanglement:

$$\begin{aligned} T_{delays} &= Twg_1 + \Delta Tw^1_{j+2,i} + \Delta Tw^2_{j+3,i+1} \\ Twg_1 &= \Delta Tc^2_{j+1,i+1} \\ \Delta Tc^2_{j+1,i+1} &= Tc^2_{j+1,i+1} + Two^2_{j+1,i} + To^2 \\ \Delta Tw^1_{j+2,i} &= Tw^1_{j+1,i} - \Delta Tc^2_{j+1,i+1} \\ \Delta Tw^2_{j+3,i+1} &= Tw^2_{j+3,i+1} - \Delta Tw^1_{j+2,i} \end{aligned} \tag{4.24}$$

The **Land time entanglement** (Figure 4.8b) arises the problem of finding out which UAV is going to arrive first at the group of vertices vg_1. In the following we explain the process to solve vg_1 and the whole sub-tour. To clarify this explanation, we will use Figure 4.10 as an example of a task plan for vg_1 in Figure 4.8b. There, we can observe that UAV$_1$ lands before UAV$_2$, and so, UAV$_2$ has to wait ($Twl^2_{j+3,i+1}$).

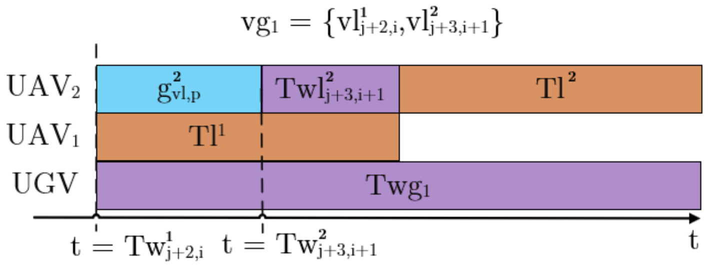


Figure 4.10: A task plan example with one UGV and two UAVs, for computing vg_1 in the Land time entanglement of Figure 4.8b. The UAV_2 has to wait for the landing with $Twl^2_{j+3,i+1} > 0$. Also, note that the total group delay $Twg_1 = Tw^2_{j+3,i+1} + Twl^2_{j+3,i+1} + Tl^2$.

First, let P^k_j denote the time spent from the first vertex of the sub-tour $vt^k_{j,i}$, to the corresponding take off vertex of the landing vertex in vg_1, i. e., $vt^2_{j+1,i+1}$ corresponds to $vl^2_{j+x,i+1}$. Let C^k_j denote the function used to sort the vertices in vg_1, and s_1 represent the ordered tuple of vertices. Then, the criterion set to sort the vertices is shown in Equation 4.25. For instance, $vg_1 = \{vl^1_{j+2,i}, vl^2_{j+3,i+1}\}$ in Figure 4.10.

$$\begin{aligned} P^k_j &= \sum_{h=1}^{H(vl^k_j)} \left(\Delta Tc^{U(h)}_h \cdot X_h + \Delta Tw^{U(h)}_h \cdot \overline{X_h}\right) + t^{h,h+1}_{ugv} \\ C^k_j &= P^k_j + t^{j,k}_{uav} - Tl^k \ \forall j \in vg_1 \\ vg_1 &= \{z \in \{1, \dots, m-1\} : C^k_z < C^k_{z+1})\} \\ s_1 &= \{vt^1_{j,i}, vt^2_{j+1,i+1}, vg_1\} \end{aligned} \tag{4.25}$$

where $H : vl^k_{j,i} \rightarrow vt^k_{j,i}\ \forall i \in \{1, \ldots, n\}$ represents the mapping function to map the landing vertices to their corresponding take off vertices, $U : U(h) \rightarrow u_k$ is the mapping function which returns the corresponding UAV u_k of vertex v_h, and X_h is a binary variable which takes a value 1 if v_h is a take off vertex or it is the last vertex of an ordered tuple vg_z; or it takes a value 0 if v_h is a land vertex or it is the last vertex of an ordered tuple vg_z.

Constraint (xi): only one UAV can land at a time

Second, this method formulates $Twl^k_j\ \forall\ j > 1 \in s_1$ to fulfil Constraint (xi) of mUCVLMP as in Equation 4.26. For instance, Figure 4.10 shows that the UAV$_2$ has to wait to land, so $Twl^2_{j+3,i+1} > 0$.

$$Q(j,k) = Tw^k_j - (Tw^k_{j-1} + Twl^k_{j-1} + Tl^k_{j-1})$$
$$Twl^k_j = \begin{cases} |Q(j,k)| & ,\text{if } Q(j,k) < 0 \\ 0 & ,\text{otherwise} \end{cases} \tag{4.26}$$

Third and last, the ordered tuple $s_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}, vg_1\}$ and the computed Twl^k_j in the previous steps enable the method to compute the group delay Twg_1. Thus, we can obtain the total rendezvous delay T_{delays} for the Land time entanglement. For instance, $Twg_1 = Tw^2_{j+3,i+1} + Twl^2_{j+3,i+1} + Tl^2$ in Figure 4.10. The following Equation 4.27 shows the formulation to compute T_{delays} in a Land time entanglement:

$$\begin{aligned} T_{delays} &= \Delta Tc^1_{j,i} + \Delta Tc^2_{j+1,i+1} + Twg_1 \\ \Delta Tc^1_{j,i} &= Tc^1_{j,i} + Two^1_{j,i} + To^1 \\ \Delta Tc^2_{j+1,i+1} &= Tc^2_{j+1,i+1} + Two^2_{j+2,i+1} + To^2 \\ Tc^2_{j+1,i+1} &= -\Delta Tc^1_{j,i} - t_{ugv}^{vt^1_{j,i}, vt^1_{j+1,i+1}} \\ Twg_1 &= Tw^2_{j+3,i+1} + Twl^2_{j+3,i+1} + Tl^2 \end{aligned} \tag{4.27}$$

The **Semi time entanglement** (Figure 4.8c) arises the problem of selecting the takes off and landings order. This is an interesting spot because it can drive to different vertices configurations for achieving an optimal task parallelization. Nevertheless, the waiting time for landing Twl^k_j of a UAV is critical, because it can lead it to run out of battery and so, a failed plan. For instance, many UAVs taking off before a UAV waiting for landing, could lead a single UAV to run out of battery because of that excessive delay.

In this way, we describe a mandatory criterion to solve this entanglement: any UAV takes off before all the landings have been completed. Following this criterion, vg_1 can be split up in two subsets: the set of take off vertices Vt, and the set of landing vertices Vl. Thus, both subsets are sorted by computing Two and Twl as in the two previous Take off and Land time entanglements. The subset Vl is firstly sorted following the Land time entanglement as denotes Equation 4.26, and secondly, Vt is sorted as in a Take off time entanglement in Equation 4.22. Naturally, Vt has to be sorted secondly, because it needs to update the rendezvous delays of the vertices in Vl, as well as the UAV's energy function. So, the ordered tuple of the sub-tour is $s_1 = sorted(Vl) \cup sorted(Vt)$, and the group delay Twg_1 of this Semi time entanglement is the last vertex of s_1. Thus, the total rendezvous delay is computed as follows:

$$\begin{aligned} T_{delays} &= \Delta Tc^1_{j,i} + Twg_1 + Tw^2_{j+3,i+1} \\ \Delta Tc^1_{j,i} &= Tc^1_{j,i} + Two^1_{j,i} + To^1 \\ Twg_1 &= \Delta Tc^2_{j+2,i+1} \\ \Delta Tc^2_{j+2,i+1} &= Tc^2_{j+2,i+1} + Two^2_{j+2,i+1} + To^2 \\ \Delta Tw^2_{j+3,i+1} &= Tw^2_{j+3,i+1} \end{aligned} \tag{4.28}$$

The **Total time entanglement** (Figure 4.8d) presents all the previously commented problems of time entanglements, where $vg_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}, vl^2_{j+2,i+1}, vl^1_{j+3,i}\}$. The criterion set to solve this entanglement follows an ordered set of steps to sort vg_1, and compute the total rendezvous delay of the sub-tour. To clarify this explanation, we will use Figure 4.11 as an example of a task plan for vg_1 in Figure 4.8d.

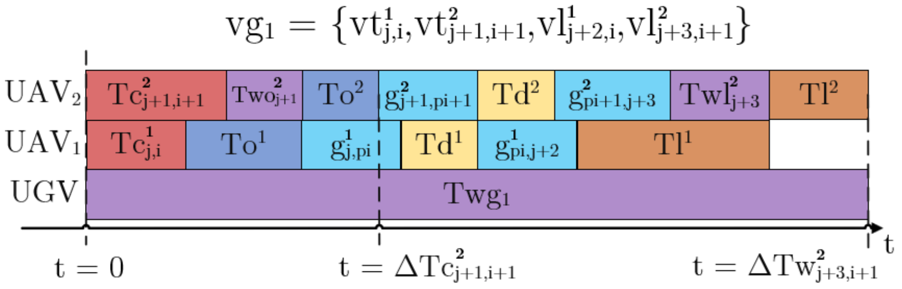


Figure 4.11: A task plan example with one UGV and two UAVs, for computing vg_1 in the Total time entanglement of Figure 4.8d. As we can appreciate, $T_{delays} = Twg_1 = \Delta Tc^2_{j+1,i+1} + \Delta Tw^2_{j+3,i+1}$.

The steps to solve this entanglement are described as follows:

1. Sort the take off vertices of vg_1 following the criterion of Equation 4.22. For instance, $vg_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}\}$ in Figure 4.11.

2. One by one, insert the next take off vertex vt and its corresponding vl in the ordered tuple of the sub-tour s_1, which implies computing the divorce and rendezvous delay of each delivery considering the criteria of Equation 4.22 and Equation 4.26 to compute Twl and Two. For instance, Figure 4.11 shows that UAV_1 performs charging ($Tc^1_{j,i}$) and takes off firstly (To^1). Then, UAV_2 has to wait for the take off (To^2), so $Two^2_{j+1} > 0$. Thus, both of them perform parallel deliveries (Td^1 and Td^2), but UAV_1 arrives

first to the UGV, so it lands (Tl^1) without waiting. Therefore, UAV$_2$ has to wait ($Twl^2_{j+3} > 0$) before landing (Tl^2) on the UGV. The sub-tour is finished when UAV$_2$ is finished landing.

3. On every vertex insertion, there arise three possible scenarios to compute the delays. The first scenario is denoted by $vg_1 = \{vt^1_{j,i}, vl^1_{j+1,i}, vt^2_{j+2,i+1}, vl^2_{j+3,i+1}\}$, so the divorce and rendezvous delay of the vertices are formulated as in the No distance entanglement (see Equation 4.17 and Equation 4.18). The second scenario is denoted by $vg_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}, vl^1_{j+2,i}, vl^2_{j+3,i+1}\}$, so they are formulated as the Semi distance entanglement (see Equation 4.19). Lastly, the third scenario is defined by $vg_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}, vl^2_{j+2,i+1}, vl^1_{j+3,i}\}$, so it is formulated as the Total distance entanglement (see Equation 4.20). For instance, the ordered tuple of the sub-tour s_1 is equal to $vg_1 = \{vt^1_{j,i}, vt^2_{j+1,i+1}, vl^1_{j+2,i}, vl^2_{j+3,i+1}\}$ in Figure 4.11, so it is formulated as a Semi distance entanglement.

Note that $s_1 = vg_1$

4. Once all the vertices have been inserted in s_1 , the total rendezvous delay T_{delays} of the sub-tour is equal to the group delay Twg_1, which is particularly computed as the total delivery time of the last delivery. For instance, $Twg_1 = \Delta Tc^2_{j+1,i+1} + \Delta Tw^2_{j+3,i+1}$ in Figure 4.11. The following Equation 4.29 shows the formulation to compute T_{delays} in a Total time entanglement:

$$\begin{aligned} T_{delays} &= Twg_1 \\ Twg_1 &= \Delta Tc^2_{j+1,i+1} + \Delta Tw^2_{j+3,i+1} \\ \Delta Tc^2_{j+i,i+1} &= Tc^2_{j,i} + Two^2_j + To^2_j \\ \Delta Tw^2_{j+3,i+1} &= g^2_{j,pi} + Td^2 + g^2_{pi,j} + Twl^2_j + Tl^2 \end{aligned} \tag{4.29}$$

Finally, we have demonstrated that the geometrical rendezvous method and the above elemental entanglements are able to represent any complex combination of vertices that may occur in the mUCVLMP defined in section 4.1. Also, we have shown that the total rendezvous delay of all of them can be formulated as an ordered function of the divorce and rendezvous delays. Therefore, following the rendezvous method equations to compute the total rendezvous delays on each elemental entanglement, we obtain the objective function 4.4 of the mUCVLMP:

$$T_{delays} = \sum_{j=1}^{m} \sum_{i=1}^{n} \sum_{k=1}^{s} \Delta Tc^k_{j,i} \cdot X_j + \Delta Tw^k_{j,i} \cdot \overline{X_j} \tag{4.30}$$

where X_j is a binary variable which takes a value 1 if $v_j \in Vt$, or if it is the last take off vertex of an ordered tuple vg_z; or it takes a value 0 if $v_j \in Vl$, or it is the last landing vertex of vg_z.

In the next section, we describe a memetic search, which integrates this method to look for the best combination of vertices and UAVs for minimizing the objective function 4.4 of the mUCVLMP.

4.2.2 *The memetic search for the delivery optimization*

The memetic search of COURIER aims to find the ordered tuple of vertices and delivery UAVs that minimize the objective function 4.1 of the mUCVLMP. Memetic algorithms (MA) were firstly introduced by Moscato et al. [254], and extended from the well-known genetic algorithms by adding a local search operator to avoid the search process a premature convergence. Our memetic search implements a search procedure to find a feasible solution to the mUCVLMP. In the following, we first describe the fitness landscape [255] of the algorithm, which is the weighted graph abstractly made by the algorithm to search for a feasible solution. And, second, we present the devised search procedure.

From now on, valid is equal to feasible

Our MA is a search algorithm focused on solving the mUCVLMP by splitting the problem into sub-tours, as shows the rendezvous method in the previous section, and solving each sub-tour individually. Each sub-tour $s_j \in S$ is represented by a vertex v_j, the two adjacent vertices $\{v_{j-1}, v_{j+1}\}$ and the area covering a subset of parcels of the problem $a \leftarrow \{p_i, \dots, p_n\}$. Then, we denote a configuration in the search algorithm, as the genetic information required to generate a valid solution for a single sub-tour. As shows Figure 4.12, each configuration is a chromosome formed by genes, where each gene has the genetic information for a specific parcel delivery: the parcel coordinates p_i, the UAV information u_k, the rendezvous vertices $\{vt_{j,i}, vl_{j+x,i}\}$, and the rendezvous delay $Tw_{\%}$ as a percentage between the minimum and maximum delay allowed for each parcel. This rendezvous delay will be used by the heuristic search to represent low and high values of the real Tw for every parcel indistinctly, but always meeting the inequality of Equation 4.7b. That is, a low value of $Tw_{\%}$ means a low value of the real Tw, and vice versa. For instance, the maximum value of $Tw_{\%}$ is 100%, so that always $vt_{j,i} = vl_{j+x,i}$. These four features enable the algorithm to fill the search space with a broad spectre of configurations with any combination of vertices and UAVs.

4.2.2.1 *The fitness landscape*

The fitness landscape of a MA is characterized by its **search space**, **neighbourhood relation** and **guiding function** [255]. We explain all three below, looking more closely at the last one because of its relevance to the algorithm.

The **search space** of our algorithm is shaped by an ocean of configurations with different vertices and UAVs combinations. It is a weighted graph, where the algorithm will dive to look for a feasible solution

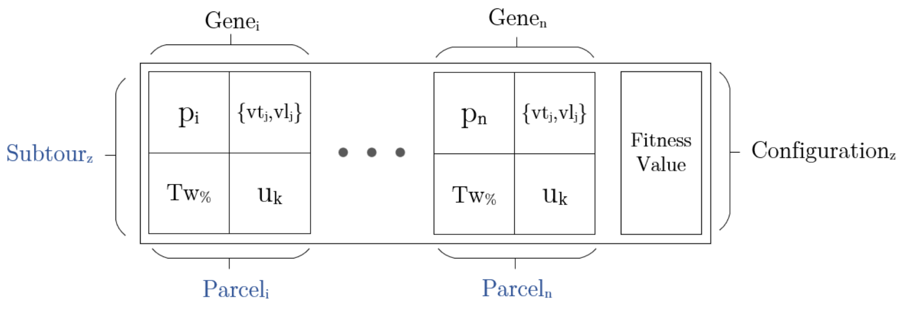


Figure 4.12: The genetic information kept in a chromosome to represent a valid configuration for a sub-tour of the mUCVLMP. Each chromosome has a set of genes, where each gene represents a parcel delivery.

for every sub-tour. As an evolutionary algorithm, our MA deploys a population-based search, and it keeps a heterogeneous group of configurations on each iteration, known as population. This is a key feature that differs them from the local search algorithms[3]. Thus, a population represents a sub-graph of the above configurations in the search space, i. e., a set of valid solutions for a single sub-tour.

The **neighbourhood relation** represents the moves that build arcs from a single configuration to its neighbours in the search graph. Our algorithm allows transitions between configurations by modifying only the $\{Tw_{\%}, u_k\}$ genetics values. However, not all transitions are allowed, because some of them can lead the algorithm to an invalid solution. Thus, a valid transition is characterized for satisfying all the problem constraints specified in section 4.1. For instance, an invalid solution could be represented by an incorrect sorting among the rendezvous vertices, i. e., the landing vertex is before the take off vertex in the UGV's path, which is forbidden, or it can be represented by a semi distance entanglement (see Figure 4.7b) where the vertices have the same UAVs, which is also forbidden by the mUCVLMP.

The **guiding function** gives a weight to each configuration in the search graph, and it guides the algorithm through the graph. In general terms, the guiding function, also known as fitness function, assesses the quality of the configuration. Figure 4.12 shows that each configuration keeps its own fitness value. The objective of our MA is to find the configuration which minimizes the fitness function. For the quality assessment of a configuration solving a sub-tour s_z, we define the following fitness function as a sum of functions measuring different factors of the objective function 4.1:

$$Fitness_z = F_{tw} + F_{\sigma} + F_{tc} + F_{free} \tag{4.31}$$

The function F_{tw} measures the quality of the rendezvous in the sub-tour, aiming to minimize the rendezvous delay Tw of every parcel.

3 Local search algorithms only keep one configuration on each iteration.

Specifically, F_{tw} computes the normalized sum of the single rendezvous Tw. So, the higher the rendezvous time the higher F_{tw}, and so, the worse the fitness value. Here, it is important to know that it does not compute any entanglement as described in the rendezvous method, otherwise, it would compute the exact solution in an enormous computational time. Then, this function computes the rendezvous of each parcel independently, as described in Equation 4.13, and also, it checks that every configuration meets the inequality of Equation 4.7b. Given the set of parcels in a sub-tour $p_i \in P$ where $i = 1 \ldots n$, the fitness value of this function is described as follows:

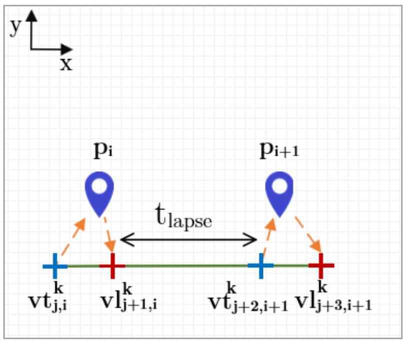
$$F_{tw} = \frac{1}{max(Tw)} \sum_{i=1}^{n} Tw_i \tag{4.32}$$

The function F_σ measures the degree of entanglements in the sub-tour, that is, it counts the number of intermediate vertices between each rendezvous pair $\{vt^k_{j,i}, vl^k_{j+x,i}\}$, for every parcel in the sub-tour. For example, the sub-tour $s_j = \{vt^k_{j,i}, vt^k_{j+1,i+1}, vl^k_{j+2,i}, vl^k_{j+3,i+1}\}$ has $F_\sigma = 2$, because each rendezvous pair has 1 intermediate vertex. Also, it checks if the configuration meets the inequality of Equation 4.7a. On the one hand, a higher value could lead the algorithm to find configurations with a high degree of entanglements, which can lead to configurations with too complex entanglements, so the UAVs reach the rendezvous vertex before the UGV, i. e., $\Delta Tw < 0$, which is forbidden. On the other hand, a lower value could give too simple configurations where there are not parallel deliveries, hence, the algorithm would lose efficiency. Given the ordered tuple of vertices $s_z = \{v_j, \ldots, v_m\}$ where $j = 1 \ldots m$, the fitness value of this function is described as follows:

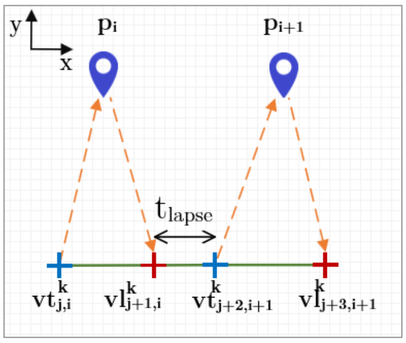
$$F_\sigma = \sum_{j=1}^{m} \Big(j - I(vl_j)\Big) \cdot X_j \tag{4.33}$$

where $I : vl \rightarrow vt$ represents the mapping function to get the index of the take off vertex vt_j linked to vl_j, and X_j is a binary variable which takes value 1 if the vertex in j is a landing vertex.

The function F_{tc} estimates the time lapse given to a UAV to perform charging in order to minimize the divorce delay ΔTc of every parcel. Firstly, it computes f_a as the sum of the inverse of the normalized squared total time between every two consecutive parcels visited for the same UAV. This is an estimation of the time that a UAV will spend charging. For instance, Figure 4.13a shows that the time lapse t_{lapse} is the UGV's travel time from $vl^1_{j+1,i}$ to $vt^1_{j+2,i+1}$. Secondly, it computes f_b as the sum of the inverse of the normalized difference between the UAV's travel time of every two consecutive parcels for the same UAV.



(a) A configuration with a low fitness value F_{tc}.



(b) A configuration with a high fitness value F_{tc}.

Figure 4.13: A comparison between a low and high fitness value of F_{tc}.

For instance, Figure 4.13a shows a configuration where the time lapse t_{lapse} between two consecutive parcels is higher than in Figure 4.13b. Also, the UAV's travel time in Figure 4.13a is lower than Figure 4.13b. Then, Figure 4.13a represents a scenario where the divorce delay in $vt^k_{j+2,i+1}$ will be minimum. However, Figure 4.13b shows that the divorce delay in $vt^k_{j+2,i+1}$ is expected to be higher. Therefore, the F_{tc} fitness value of the configuration in Figure 4.13a will be lower than the F_{tc} of the configuration in Figure 4.13b, because the divorce delay in $vt_{j+2,i+1}$ is estimated to be higher in the second case. Then, given the set of UAVs U where $k = 1 \dots s$, the fitness value of this function F_{tc} is described as follows:

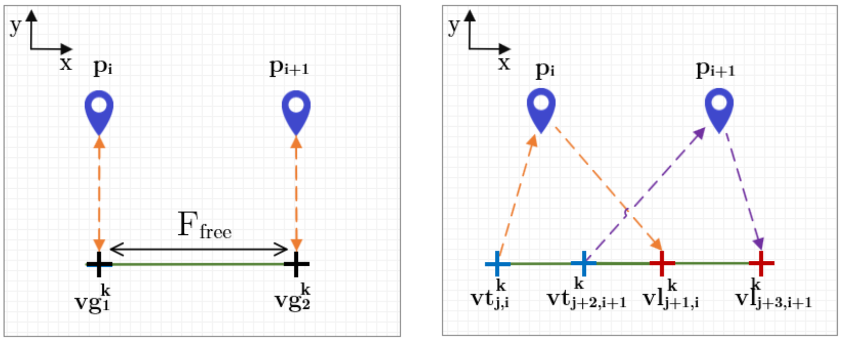
$$\begin{aligned} F_{tc} &= f_a + f_b \\ f_a &= \sum_{k=1}^{s} 1 - \left(\frac{t^k_{lapse}}{max(t_{lapse})} \right)^2 \\ f_b &= \sum_{k=1}^{s} 1 - \left(\frac{T^k_{end} - T^{p_i}_{trip} + T^k_{end} - T^{p_{i+1}}_{trip}}{T^k_{end}} \right) \end{aligned} \tag{4.34}$$

where t^k_{lapse} is the total time lapse between every two consecutive parcels delivery by the UAV k, T^k_{end} is the UAV's endurance in time terms, and $T^{p_i}_{trip}$ is the total travel time to deliver the parcel p_i.

The function F_{free} measures the total time in which no delivery is being carried out, also called free time. That is, it computes the total free time in order to look for solutions with a high degree of parallel working or minimum free time. Given the ordered tuple of vertices $s_z = \{v_j, \dots, v_m\}$ where $j = 1 \dots m$, the fitness value of F_{free} is computed as follows:

$$F_{free} = t^{0,1}_{ugv} + \sum_{j=1}^{m-1} \left(t^{j,j+1}_{ugv} \cdot X_j \right) + t^{m,0}_{ugv} \tag{4.35}$$

where X_j is a binary variable which takes value 1 if j is a landing vertex and $j+1$ is a take off vertex, and 0 otherwise. Also, $t_{ugv}^{0,1}$ is the UGV's path from the depot to the first vertex, and $t_{ugv}^{m,0}$ denotes the return to the depot from the last vertex. Both of them are essential to compute the total free time along the UGV's path.



(a) A configuration with a high fitness value F_{free}.

(b) A configuration with a fitness value $F_{free} = 0$.

Figure 4.14: A comparison between a low and high fitness value of F_{free}.

Note that F_{free} is completely different to t_{lapse}^k computed in F_{tc}, because the latter concerns to each UAV individually, whereas F_{free} is the time when no UAV is flying. This is a small but important difference, because t_{lapse}^k allows the algorithm to look for solutions with minimum divorce delays ΔTc, and F_{free} to select those solutions with less free time possible, and a minimum impact in the divorce delays. For instance, Figure 4.14a shows a scenario with a F_{free} higher value than Figure 4.14b, where $F_{free} = 0$. Here, the fitness will guide the algorithm through solutions like Figure 4.14b.

Once the fitness landscape has been described as a weighted graph with all the feasible and infeasible solutions to the mUCVLMP, we now present the search procedure devised to carry out an efficient navigation through the weighted graph.

4.2.2.2 *The search procedure*

The search procedure is composed by a sequence of architectural components strategically placed to enable an efficient and agile navigation in the weighted search graph. Algorithm 8 presents an overview of the main components that form the search procedure. It is basically the standard scheme for MA devised by Moscato and Cotta [256]. However, there is a wide spectre of strategies and operators that can be combined in countless ways depending on the problem. As Figure 4.4 shows, the memetic search gets as inputs the set of areas A', the set of vertices V'', the radius R, the UAV's data and the UGV's data. Then, these inputs are split up into sub-tours $s_j \in S$, with which the search procedure in Algorithm 8 works. In the following, we explain the implemented seven stages strategy, where each stage is performed by a particular function of Algorithm 8.

Algorithm 8 The Memetic Search of COURIER

```
1: Procedure SearchProcedure (lastsubtour,subtour,uavs,ugv)
2: geoData ← computeGeoFeatures(subtour)
3: population ← genInitPop(lastsubtour,subtour,geoData,uavs,ugv)
4: while not terminationCriterion() do
5:    newPopulation ← generateNewPopulation(population)
6:    population ← updatePopulation(population,newPopulation)
7:    if hasDegenerated(population) then
8:       population ← restartPopulation(population)
9:    end if
10: end while
11: End procedure
```

FIRST STAGE The *computeGeoFeatures* function in line 2. This function is specifically placed at the beginning to compute the three geometrical features of each parcel (see Figure 4.5), required to implement the geometrical rendezvous method presented in subsection 4.2.1.

SECOND STAGE The *genInitPop* function in line 3. This function is a cornerstone in the search procedure, as constructs an initial population with configurations that place the search to the right space of solutions. Due to the massive search graph that might be created because of the combinations of vertices, we embed a constructive procedure (see Algorithm 9) to inject valid or nearly valid configurations into the initial population.

Algorithm 9 A constructive procedure for the initial population

```
1: Procedure genInitPop(subtour,geoData,UAVs)
2: initTws ← initializeTws(subtour,geoData)
3: slowerUAV ← getSlower(UAVs)
4: for $p_i \in P$ do
5:    $fuav^i$ ← interpolateTuav($geoData^i$,slowerUAV)
6:    $radius^i$ ← interpolateRadius($geoData^i$,$tuav^i$,initTws)
7:    $rendez^i$ ← interpolateRendez(subtour,$geoData^i$,$radius^i$)
8: end for
9: v_combinations ← computeCombinations(rendez)
10: configurations ← buildConfigurations(v_combinations,UAVs)
11: initPopulation ← computeFitness(configurations)
12: return initPopulation
13: End procedure
```

This constructive procedure initializes values of the genetic feature $Tw_{\%}$ to interpolate its corresponding rendezvous vertices, and so, generate a vertices combination matrix with all the configurations. That is, each $Tw_{\%}$ represents a unique rendezvous for every parcel delivery (see Figure 4.15). Depending on the number of targets in the sub-tour, this procedure starts selecting more or less initial values of $Tw_{\%}$ (*intializeTws* function in line 2), always starting from $Tw_{\%} = 100$. Note that $Tw_{\%} = 100$ represents the easiest rendezvous because $vt_{j,i} = vl_{j+x,i}$, but it is not usually a good configuration to minimize mUCVLMP.

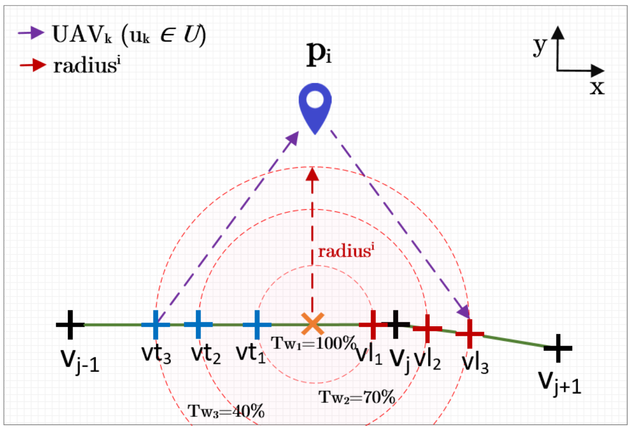


Figure 4.15: A graphical representation of the constructive procedure implemented in the memetic algorithm. Each $Tw_{\%}$ represents a particular radius with the orthogonal point as the center. So, $Tw_{\%} = 100\%$ in the orthogonal point. Also, we assume that $radius_i \simeq d_{vt_{j,i},vl_{j+x,i}}/2$.

This procedure implements the three following functions to compute the rendezvous vertices of each $Tw_{\%}$:

1. The *interpolateTuav* function in line 5. It estimates the UAV's travel time $t_{uav}^{j,i,k}$ for each parcel depending on the Euclidean distance (instead of the distance along the UGV's path) between the take off and the landing vertices $d(vt_{j,i}, vl_{j+x,i})$. Thus, the independent variable of $fuav^i$ is the distance between the two rendezvous vertices (take off and landing), and the dependent variable is the UAV's travel time. The two vertices represent an estimation of the maximum and minimum $fuav^i$ required for that parcel delivery. Here, we assume the following linear correlation: the higher $d(vt_{j,i}, vl_{j+x,i})$, the higher $fuav^i$. Then, the minimum $fuav^i$ is computed with $min(d(vt_{j,i}, vl_{j+x,i}))$, which is usually when the rendezvous vertices are both the orthogonal point of that parcel (see Figure 4.5), so $d(vt_{j,i}, vl_{j+x,i}) = 0$. The maximum $fuav^i$ is computed with $max(d(vt_{j,i}, vl_{j+x,i}))$, which is usually the distance between the two farthest vertices in the sub-tour. One vertex is always the last vertex of the sub-tour (see v_{j+1} in Figure 4.15), and the other vertex sometimes is the first vertex of the sub-tour (see v_{j-1} in Figure 4.15) or the last landing

vertex of the previous solved sub-tour in the problem. These maximum and minimum values interpolate the $fuav^i$ function.

2. The *interpolateRadius* function in line 6. It computes the maximum and minimum rendezvous delays for each parcel, so that, it can interpolate the radius of a particular $Tw_{\%}$ as Figure 4.15 shows. For that, we make the following assumption for estimating $t_{ugv}^{j,i}$:

$$Tw_{j,i}^{k} = t_{uav}^{j,i,k} - t_{ugv}^{j,i} \tag{4.36}$$

$$t_{ugv}^{j,i} \simeq \frac{radius_i}{V_{ugv}} \simeq \frac{2 \cdot d(vt_{j,i}, vl_{j+x,i})}{V_{ugv}} \tag{4.37}$$

where $d(vt_{j,i}, vl_{j+x,i})$ is the Euclidean distance. Then, we make the following system of equations to compute the rendezvous delay with maximum radius:

$$\left.\begin{aligned} \frac{2 \cdot max(d(vt_{j,i}, vl_{j,i}))}{V_{ugv}} + Tw_{j,i}^{k} = f_{uav}^{j,i,k} \\ f_{uav}^{j,i,k} = f_{uav}^{j,i,k}(max(radius_i)) \end{aligned}\right\} \tag{4.38}$$

Also, the rendezvous delay with minimum radius is easily computed with $d(vt_{j,i}, vl_{j+x,i}) = 0$, where the UAV's travel time is equal to the total time the UGV has to wait to the UAV to deliver a parcel, i. e., $f_{uav} = Tw_{j,i}^{k}$. Once these two limits are computed, the procedure gets the particular *radius*^{i} of each *initTws* gathered at the beginning.

3. The *interpolateRendez* function in line 7. It uses the *radius*^{i} values to compute the rendezvous vertices as shown in Figure 4.15. For that, it builds the following system of equations:

$$\left.\begin{aligned} f_j(x)\ \forall j \in s_j \\ (x - x_{port})^2 + (y - y_{port})^2 = radius_i^2 \end{aligned}\right\} \tag{4.39}$$

where $f_j(x)$ is the line of the UGV's directed path in sub-tour s_j (see Figure 4.5), (x_{port}, y_{port}) are the Cartesian coordinates of the orthogonal point, and, (x, y) are the Cartesian coordinates of the rendezvous vertices. Note that each $radius_i$ give us two vertices: the first one in the UGV's directed path is the take off vertex, and the second one is the landing vertex.

Once the rendezvous vertices of each initial *Tw* have been computed, the constructive procedure of Algorithm 9 builds a matrix with all the possible combinations of vertices for each parcel[4] (*computeCombinations* function in line 9). Then, it assigns a random UAV to each gene to complete the configurations (*buildConfigurations* function in line 10). Then, this constructive procedure finishes by computing the fitness of each configuration following the Equation 4.31, and selecting the best feasible configurations to construct the initial population (*computeFitness* function in line 11).

THIRD STAGE The *terminationCriterion* function in line 4. It checks if any of the following stop conditions are met: size of the population below than a threshold, number of restarts over a threshold, total number of iterations reached and/or the convergence of the population.

FOURTH STAGE The *generateNewPopulation* function in line 5. It defines the procedure of Algorithm 10 to bring up a new population in the search process. It implements a mixed strategy as a sequence of four phases, where the first phase is done individually, and the other three are carried out in an iterative process. The four phases are: (i) Heuristic recombination (lines 5-9), (ii) Blind recombination (lines 10-15), (iii) Mutation (lines 17-22) and (iv) Local-search (lines 24-27).

i. Heuristic recombination (lines 5-9). We devise a heuristic recombination operator to select the parental features that will be transmitted to the next population. On each iteration, it selects the best two parents of a sub-group of the current population (*tournamentSelection* in line 6), and then, it applies the heuristic recombination operator. We have implemented the dynastically optimal recombination operator [257] (*dynasticalRecombination* in line 7), whose objective is to create all the possible descendants of the selected parents, and to find the best configuration of the parental features. The recombination of the parental features is done at the level of gene. That is, the genetic features of a $gene_i$ of the parent configuration A can only be recombined with the genetic features of the same $gene_i$ of the parent configuration B. This is because a specific value of $Tw_{\%}$ might not represent the same for every parcel in the sub-tour, so the algorithm would have to compute the rendezvous vertices for that value as shown in *genInitPop* procedure. Thus, this will result in an explosion of the computational time of the algorithm. Therefore, we mix the genetic features among the same genes, so the algorithm does not have to compute the rendezvous vertices again. Also, this stage has been placed out of the iterative process to avoid the following phases disturbing the smart element inserted in this heuristic stage.

4 The number of combinations is computed as $m^{2 \cdot p}$, where m is the number of initial Tws values for each parcel, and p is the number of parcels.

Algorithm 10 A mixed strategy for bringing up a new population

```
1: Procedure generateNewPopulation(population)
2: newPopulation ← ∅
3: blindPopulation ← ∅
4: mutPopulation ← ∅
5: for $i \in n_h$ do //heuristic recombination
6:     parents ← tournamentSelection(population)
7:     descendants ← dynasticalRecombination(parents)
8:     newPopulation ← newPopulation ∪ descendants
9: end for
10: for $i \in n_b$ do //blind recombination
11:     parents ← stochasticSelection(population)
12:     descendants ← blindRecombination(parents)
13:     blindPopulation ← blindPopulation ∪ descendants
14: end for
15: newPopulation ← newPopulation ∪ blindPopulation
16: if blindPopulation ≠ ∅ then
17:     for $i \in n_m$ do //mutation
18:         winner ← tournamentSelection(blindPopulation)
19:         newMutation ← mutation(winner)
20:         mutPopulation ← mutPopulation ∪ newMutation
21:     end for
22:     newPopulation ← newPopulation ∪ mutPopulation
23: end if
24: if mutPopulation ≠ ∅ then //local search
25:     LSPopulation ← localSearchEngine(mutPopulation)
26:     newPopulation ← newPopulation ∪ LSPopulation
27: end if
28: return newPopulation
29: End procedure
```

ii. Blind recombination (lines 10-15). We allocate a blind recombination operator which does not use any heuristic information to produce the descendants. On each iteration, it selects two parents through a stochastic method (*stochasticSelection* in line 11), and then, it calls the recombination operator. We have implemented a uniform crossover, whose objective is to generate two descendants from the crossing between the two parents at randomly selected crossing points. As well as the heuristic operator, the blind recombination is performed at the level of gene.

iii. Mutation (lines 17-22). The descendants generated by the previous blind recombination operator are put on to a mutation process. There is a tournament to select the best configuration among a sub-group, and then, the genetic features $Tw_{\%}$ and u_k are mutated. The mutation procedure randomly selects the genes that are going to be mutated, and then, it randomly selects the genetic features to mutate. The mutation of the UAV genetic feature is done by randomly selecting a different UAV. Also, the mutation of the $Tw_{\%}$ genetic feature requires to compute the rendezvous vertices for the new value.

iv. Local-search (lines 24-27). The mutated population is driven to a local search engine. We have designed a stratified approach [258] in which the mutated population is split up into *n* levels, and then, a tournament selection gets the best individual of a sub-group of every level. The selected individuals are then put on a hill-climbing search to minimize their fitness function. This search stops when the fitness function has been minimized, or a limit of iterations has been reached, which prevents the algorithm to turn a simple solution into non-polynomially solvable.

FIFTH STAGE The *updatePopulation* in line 6. This function rebuilds the old population with the new computed population. In the literature, there are several strategies [259] to build the new population, but we have implemented the *plus strategy*. This strategy constructs the new population by taking the best configurations from the $population \cup newPopulation$.

SIXTH STAGE The *hasDegenerated* function in line 7. This function checks if the search procedure has degenerated. For this assessment, we implement a procedure following the Shannon's entropy concept [260] to study the diversity of configurations in the population. That is, a population is degenerated if the number of equal configurations is higher than a threshold. In this way, the search procedure has degenerated if the number of successive degenerated populations is higher than a given threshold.

SEVENTH STAGE The *restartPopulation* function in line 8. This function resets the population in case of a degenerated search. As well, there are different strategies to build a new population considering the degenerated one. We implement the random immigrant strategy [261], which consists on selecting a percentage of the best configurations of the degenerated population, and then, add new generated solutions from the scratch. These new solutions are computed as shows the *genInitPop* procedure of Algorithm 9.

The search procedure iterates until the termination criteria is met. At this point, the memetic search of COURIER has only estimated the quality of the selected solution. Then, COURIER calls the arithmetic solver described in the next section. This solver computes the real task planning of the best solution selected as Figure 4.4 shows.

4.2.3 *The arithmetic solver for the task planning*

The arithmetic solver of COURIER implements the rendezvous method to compute the real task planning of the solution given by the memetic search. It solves each sub-tour individually, but it considers the system status after the previous sub-tour. The system status refers to the battery level of the UAVs (Tr), the divorce and rendezvous delays of the vertices (ΔTc and ΔTw respectively), and the UGV's location. In this way, solving the last sub-tour, gives the final system status and so, the task planning completes the resolution of the mUCVLMP.

The arithmetic solver (see Algorithm 11) follows an iterative execution, where it computes the system status on every vertex. As the geometrical rendezvous method shows (see subsection 4.2.1), this algorithm updates the system status depending on the vertex and the vertices entanglement. In the following, we describe the execution flow of the algorithm, defining how the system status is updated.

It starts by updating the total rendezvous delay t_{delays} and the UAV's battery level (*updateBatteries* function in line 3) to the system status of the previous sub-tour. Also, it initializes the task planning variable *plan*, which contains the sorted list of tasks for the robotic system, and the set of group delays Twg, which represents the set of all the group delays Twg_z computed in the sub-tour. Then, it performs an iterative execution in the set of vertices of the current sub-tour $currSubtour.V$, where each vertex consists in the calculus of its associated t_{delay}, the UGV's travel time $t_{ugv}^{j,j+1}$, and the updating of the UAVs' batteries.

Every vertex starts by computing the UGV's travel time to the next vertex (*groundTravelTime* function in line 8). As we explain in the rendezvous method, this time can be used by the docked UAVs to charge theirs batteries. Then, the solver makes a distinction (conditional statement in line 9) between the isolated vertices in a particular coordinate, and the group of vertices which shares the same location. Following the rendezvous method, single vertices are computed according to the formulation for the distance entanglements. Instead, the vertices at the same location are computed according to the time entanglements formulation. The conditional statement in line 9 allows the solver to know if v_j is in some subset $vg_z \in Vg$, where Vg is the set of subsets that are time entanglements in the sub-tour. As a part of a distance entanglement, the single vertex is already sorted, so the solver adds the vertex to the task planning right away. Then, the solver makes another distinction (conditional statement in line 11) to know if the single vertex is a take off vertex, i. e., $v_j \in Vt$, where Vt is the set of take off vertices in the sub-tour, or if it is a landing vertex, i. e., $v_j \in Vl$, where Vl is the set of landing vertices.

Algorithm 11 The arithmetic solver of COURIER

```
1: Procedure arithmeticSolver($lastSubtour,currSubtour,uavs$)
2: $T_{delays} \leftarrow lastSubtour.T_{delay}$
3: $uavs \leftarrow$ updateBatteries($lastSubtour,uavs$)
4: $plan \leftarrow []$
5: $Twgs \leftarrow \emptyset$
6: for $v_j \in currSubtour.V$ do
7:     $t_{delay} \leftarrow 0$
8:     $t_{ugv}^{j,j+1} \leftarrow$ groundTravelTime($v_j, v_{j+1}$)
9:     if $v_j \notin vg_z \in Vg$ then
10:        plan.add($v_j$)
11:        if $v_j \in Vt$ then
12:            $Tc \leftarrow$ computeTc($currSubtour,uavs,v_j$)
13:            $Two \leftarrow 0$
14:            $\Delta Tc \leftarrow Tc + Two + uavs(v_j).To$
15:            $t_{delay} \leftarrow \Delta Tc$
16:            $uavs \leftarrow$ updateBatteriesAfterTo($uavs,Tc,t_{ugv}^{j,j+1}$)
17:        else
18:            $\Delta Tw \leftarrow$ computeTw($currSubtour,uavs,v_j,Twgs$)
19:            $t_{delay} \leftarrow \Delta Tw$
20:            $uavs \leftarrow$ updateBatteriesAfterTl($uavs,\Delta Tw,t_{ugv}^{j,j+1}$)
21:        end if
22:    else
23:        if isFirst($v_j,Vg$) then
24:            $V_{groups} \leftarrow$ getElementalTimeEntanglements($Vg$)
25:            if $|V_{groups}[1]| > 0$ then
26:                $vg_z,uavs \leftarrow$ solveLandEnt($V_{groups}[1],plan,uavs,Twgs$)
27:            end if
28:            if $|V_{groups}[2]| > 0$ then
29:                $vg_z,uavs \leftarrow$ solveTotalEnt($V_{groups}[2]$,plan,$vg_z,uavs,Twgs$)
30:            end if
31:            if $|V_{groups}[3]| > 0$ then
32:                $vg_z,uavs \leftarrow$ solveTakeOffEnt($V_{groups}[3]$,plan,$vg_z,uavs$)
33:            end if
34:            $Twgs(z) \leftarrow$ getGroupDelay($vg_z$)
35:            $t_{delay} \leftarrow Twgs(z)$
36:            $plan$.add($vg_z$)
37:            $uavs \leftarrow$ updateBatteriesToEndGroup($uavs,vg_z,t_{delay}$)
38:        end if
39:        $uavs \leftarrow$ updateBatteriesToNextVertex($uavs,t_{ugv}^{j,j+1}$)
40:    end if
41:    $T_{delays} \leftarrow T_{delays} + t_{delay} + t_{ugv}^{j,j+1}$
42: end for
43: $plan \leftarrow lastSubtour.plan \cup plan$
44: return $plan,T_{delays}$
45: End procedure
```

On the one hand, the take off vertices are characterized by the divorce delay ΔTc, as shown in the rendezvous method. The charging time Tc depends on the current battery level Tr in time terms and the UAV's travel time T_{trip} required to deliver the parcel (see section 4.1). Thus, Tc is computed as follows (see *computeTc* function in line 12):

$$Tc = \begin{cases} 0 & , Tr > T_{trip} \\ T_{trip} - Tr & , \text{otherwise} \end{cases} \tag{4.40}$$

Therefore, the divorce delay can be computed as: $\Delta Tc = Tc + Two + To$, where $Two = 0$ because it is a single vertex. Thus, the last step is to update the battery level Tr of every UAV in the system after the take off operation (*updateBatteriesAfterTo* function in line 16). The battery level of the UAV that takes off in this vertex is computed as follows:

$$Tr^k = Tr^k + Tc^k - To^k - t_{ugv}^{j,j+1} \tag{4.41}$$

Then, the battery level of the rest of UAVs is:

$$Tr^k = \begin{cases} Tr^k - Tc^k - To^k - t_{ugv}^{j,j+1} & , \textit{if is flying } k \\ Tr^k + Tc^k + To^k + t_{ugv}^{j,j+1} & , \text{otherwise} \end{cases} \tag{4.42}$$

On the other hand, the landing vertices are characterized by the rendezvous delay ΔTw. No matter what kind of complex entanglements are involved, the *computeTw* function in line 18 implements the formulation to solve them as a composition of the described elemental distance entanglements. As well as the take off vertices, the last step is to update the battery level of every UAV (*updateBatteriesAfterTl* function in line 16). So, the battery level of the UAV that lands in this vertex is computed as:

$$Tr^k = Tr^k - \Delta Tw + t_{ugv}^{j,j+1} \tag{4.43}$$

And the battery level of the rest of UAVs is:

$$Tr^k = \begin{cases} Tr^k - \Delta Tw - t_{ugv}^{j,j+1} & , \textit{if is flying } k \\ Tr^k + \Delta Tw + t_{ugv}^{j,j+1} & , \text{otherwise} \end{cases} \tag{4.44}$$

So far, we have described how the solver computes the distance entanglements defined in subsection 4.2.1, and how it updates the battery level of every UAV. Nevertheless, if the solver detects that the current vertex belongs to a group of vertices $v_j \in vg_z \in Vg$ (*isFirst* function in line 23), the algorithm starts a procedure to solve the group

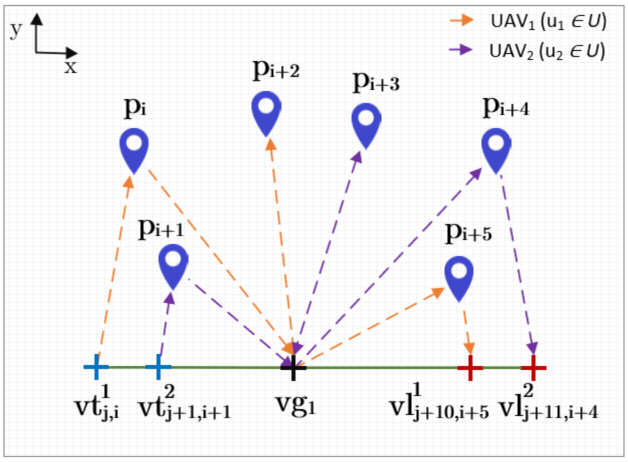


Figure 4.16: An example of a sub-tour with a complex group vg_1 formed by all the time entanglements shown in subsection 4.2.1. The sub-tour is based on six deliveries to be performed by two UAVs.

of vertices as a whole, i. e., to sort all the vertices in the group and to compute the group delay Twg_z. To do so, it splits the complex group in the elemental time entanglements (*getElementalTimeEntanglements* function in line 24): the Take off time, Land time, Semi time, and Total time entanglement. As explained in the geometrical rendezvous method, the Semi time entanglement can be computed as the union between the Take off and Land entanglements. Instead, the solver devises a particular function to solve the rest in a particular manner. The way in which the elemental time entanglements are computed, represents a specific paradigm to solve the problem. Figure 4.16 shows an instance where there are all the elemental time entanglements in a single group vg_1. We take this example to explain the paradigm for solving complex time entanglements.

Once the complex group vg_1 has been split into elemental time entanglements, we can distinguish the following three sub-groups: (1) $V_{groups}[1] = \{vl^1_{j,i}, vl^2_{j,i+1}\}$ as the Land time entanglement, (2) $V_{groups}[2] = \{vt^1_{j,i+2}, vl^1_{j,i+2}, vt^2_{j,i+3}, vl^2_{j,i+3}\}$ as the Total time entanglement, and (3) $V_{groups}[3] = \{vt^2_{j,i+4}, vt^1_{j,i+5}\}$ as the Take off entanglement. The algorithm solves these groups in an orderly manner:

1. The $V_{groups}[1] = \{vl^1_{j,i}, vl^2_{j,i+1}\}$. It computes $V_{groups}[1]$ because our top priority is to land every UAV waiting for landing, and so, avoid them to run out of battery. The *solveLandEnt* function in line 26 implements the formulation described for the Land time entanglement in order to sort the vertices and compute their rendezvous delays. Then, the solver updates the battery level of

the UAVs that are participated in this group. The batteries are updated as follows:

$$Tr^k = Tr^k - \Delta Tw \tag{4.45}$$

A possible solution for the $V_{groups}[1]$ in the vg_1 of Figure 4.16 is shown in Figure 4.17. There, we can observe that UAV$_1$ arrives and lands first on the UGV (Tl^1). Thus, the UAV$_2$ has to wait for the landing with $Twl^2_{j+3} > 0$. At this point, the task plan contains only the landing of these two UAVs, after the delivery of the parcels p_i and p_{i+1} of Figure 4.16.

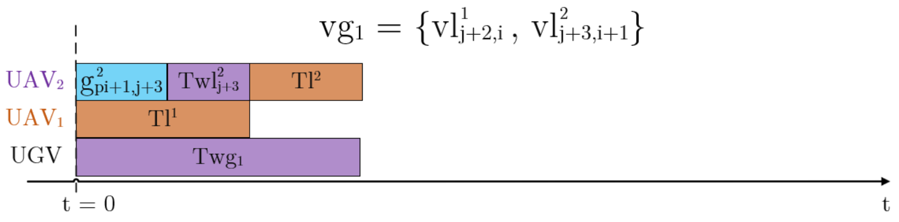


Figure 4.17: A possible task plan for the land time entanglement in $V_{groups}[1]$ of the vg_1 of Figure 4.16. It contains the landing of two UAVs, after the delivery of the parcels p_i and p_{i+1}. As the first group, vg_1 only has the sorted vertices of $V_{groups}[1]$.

2. The $V_{groups}[2] = \{vt^1_{j,i+2}, vl^1_{j,i+2}, vt^2_{j,i+3}, vl^2_{j,i+3}\}$. The deliveries in $V_{groups}[2]$ usually represent the larger delays, so the UGV has to wait to the deliveries in vg_z. Thus, they need to be performed as soon as possible. The *solveTotalEnt* function in line 29 integrates the procedure and formulation explained for the Total time entanglement in the rendezvous method. As well, this procedure gives a sorted set of vertices vg_z with their divorce and rendezvous delays. Also, it updates the batteries of the UAVs involved in $V_{groups}[2]$ as follows:

$$\begin{aligned} Tr^k &= Tr^k + Tc^k - T_{trip} \\ T_{trip} &= Two^k + To^k + g_{vt,p} + Td^k + g_{p,vl} + Twl^k + Tl^k \end{aligned} \tag{4.46}$$

A possible solution for the $V_{groups}[2]$ in the vg_1 of Figure 4.16 is shown in Figure 4.18. There, we can observe that UAV$_1$ delivers its parcel (Td^1) before the UAV$_2$ (Td^2), and so, it arrives and lands first on the UGV. At this point, the task plan contains the previous tasks, plus the complete deliveries of the parcels p_{i+2} and p_{i+3} of Figure 4.16.

It is important to highlight that the first two functions (*solveLandEnt* and *solveTotalEnt*) have as inputs the set of previous

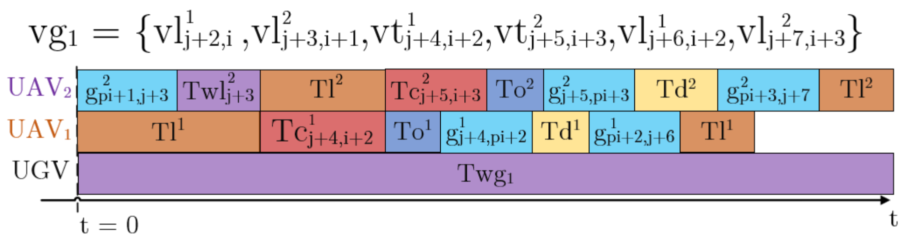


Figure 4.18: A possible task plan for the Total time entanglement in $V_{groups}[2]$ of the vg_1 of Figure 4.16. It contains the previous tasks plus the complete deliveries of the parcels p_{i+2} and p_{i+3}. As the second group, the solver adds to vg_1 the sorted vertices of $V_{groups}[2]$.

group delays $Twgs$, because they need them to update the rendezvous delay ΔTw of the landing vertices involved. Instead, the third function only has take off vertices, therefore it does not need those times.

3. The $V_{groups}[3] = \{vt^2_{j,i+4}, vt^1_{j,i+5}\}$. The last group $V_{groups}[3]$ is sorted by the *solveTakeOffEnt* function in line 32. As well as the others, this function implements the procedure and formulation to solve the Take off time entanglement described in the geometrical rendezvous method. This procedure is able to sort the vertices by computing the divorce delays of the take off vertices in $V_{groups}[3]$. Also, the batteries of the involved UAVs are updated as follows:

$$Tr^k = Tr^k + Tc^k - Two^k - To^k \tag{4.47}$$

A possible solution for the $V_{groups}[3]$ in the vg_1 of Figure 4.16 is shown in Figure 4.19. There, the takes off to deliver the parcels p_{i+4} and p_{i+5} have been added at the end of the previous task plan. We can observe that UAV$_2$ takes off first (second To^2), and so, the UAV$_1$ has to wait for its taking off (second To^1) with Two^1_{j+9}.

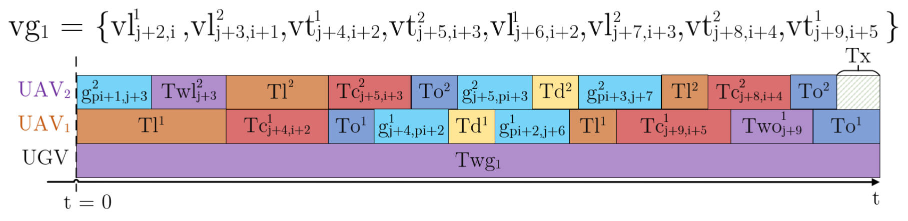


Figure 4.19: A possible task plan for the Take off time entanglement in $V_{groups}[3]$ of the vg_1 of Figure 4.16. It adds to the previous plan the task plan resulting of the sorted vertices of $V_{groups}[3]$. The green scratched square Tx represents the remaining time (not a task) to update the battery of the UAV_2 in the next step.

Once the group of vertices vg_1 has been sorted and all the divorce and rendezvous delays have been computed, the solver gets the total delay of the group Twg_1 (*getGroupDelay* function in line 34) focusing on the task plan of the UAV of the last vertex, i. e., the task plan of UAV$_1$ in Figure 4.19. Thus, $Twg_1 = Tl^1 + Tc^1_{j+4,i+2} + To^1 + g^1_{j+4,pi+2} + Td^1 + g^1_{pi+2,j+6} + Tl^1 + Tc^1_{j+9,i+5} + Two^1_{j+9} + To^1$ as shows Figure 4.19.

Also, the solver adds the task planning of the group to the general plan, and it updates the batteries of all the UAVs (*updateBatteriesToEndGroup* function in line 37) until the last task of the group (second To^1 in Figure 4.19). For the UAVs not involved in the group, their batteries are computed taking Twg_1 as the time that these UAVs can use for charging, i. e., $Tr^k = Tr^k + Twg_1$, without exceeding the battery capacity. Instead, the solver needs to compute each involved UAV individually. This is done by getting the last appearance of the UAV in the group and then, updating until the last task. For instance, Figure 4.19 shows that the UAV$_1$ does not need to update its battery, because it performs the last task of the plan (second To^1), and so, its battery was already computed by the *solveTakeOffEnt* function. However, UAV$_2$ requires to update its battery until this last task. The green scratched square Tx in Figure 4.19 shows the additional time that UAV$_2$ needs to update. The *updateBatteriesToEndGroup* makes the following calculation to update the battery of the UAV_2:

$$Tr^2 = Tr^2 - Tx \tag{4.48}$$

$$\begin{aligned} Tx = Twg_1 - g^2_{pi+1,j+3} - Twl^2_{j+3} - Tl^2 - Tc^2_{j+5,i+3} - To^2 - \\ g^2_{j+5,pi+3} - Td^2 - g^2_{pi+3,j+7} - Tl^2 - Tc^2_{j+8,i+4} - To^2 \end{aligned} \tag{4.49}$$

Then, the next step is to update the batteries of all the UAVs till the next vertex of the sub-tour (*updateBatteriesToNextVertex* function in line 39) as in Equation 4.50. Note that for every vertex in the sorted group of vertices $vg_z \in Vg$, $t^{j,j+1}_{ugv} = 0$, but for the last vertex is $t^{j,j+1}_{ugv} > 0$.

$$Tr^k = \begin{cases} Tr^k - t^{j,j+1}_{ugv} & \text{, \textit{if is flying k}} \\ Tr^k + t^{j,j+1}_{ugv} & \text{, otherwise} \end{cases} \tag{4.50}$$

Lastly, whether the solver has computed a vertex or a group of vertices, it adds the compute delay t_{delay} and the UGV's travel time $t^{j,j+1}_{ugv}$ to the total rendezvous delay T_{delays}. Then, the solver finishes its execution when the last vertex or group of vertices has been computed. Here, the solver knows the total time required to perform all the deliveries involved in the current sub-tour, and it appends the current plan to the total task planning for solving a particular mUCVLMP instance.

4.3 EXPERIMENTAL EVALUATION

In this section we present the COURIER assessment solving the mUCVLMP. We first provide an assessment analysis to characterize COURIER considering the main parameters fluctuations. This analysis explores the behaviour of COURIER considering different number of UAVs for different number of parcels. Also, this analysis investigates the impact of the UAV's endurance and UAV's speed in different configurations. Thus, the objective of this experiment is to provide an accurate algorithm characterization in a broad spectre of mUCVLMP instances. Then, a second analysis is conducted to evaluate the performance of COURIER against the heuristic approach developed in Murray and Raj [179] to solve the mFSTSP formulated by them in that research work. The objective is to evaluate the performance of the best configurations of COURIER found in the characterization analysis. Due to both mUCVLMP and mFSTSP can be framed as a generalization of the last-mile delivery problem, we design a particular experimental setup aiming to conduct a fair comparison. Also, Appendix C shows the instance generator developed for this experiment, and the parameter tuning performed to set up the memetic algorithm for a proper evaluation.

The algorithm has been implemented in MATLAB, and the experimental evaluation was carried out on a 2.6 GHz Intel Core i7 with 16 GB of RAM under Windows 10.

4.3.1 *Experiment design*

We mentioned in the literature that the last-mile delivery problem is being studied in several ways in the past decade. Due to the complex nature of the problem, there are (and keeps emerging) several formulations dealing with different constraints aiming to optimize objectives like the time, price or energy. In spite of having different objective functions, all of them can be represented as a generalization of the last-mile delivery problem. Here, we want to highlight the mFSTSP MILP formulation by Murray and Raj [179], which arises from the well-known FSTSP [4]. The mFSTSP deploys a truck with multiple UAVs to perform the deliveries aiming to minimize the travel time as well as the mUCVLMP. Also, Murray and Raj find the optimal solution for some problem instances. Thus, we selected their heuristic solution to measure the performance of COURIER, but always considering the similarities and dissimilarities of the problems, i. e., FSTSP and mUCVLMP respectively. In subsection 4.3.3, we make a further theoretical comparison before carrying out the performance experiment. Then, the following experimental setup has been designed for both the characterization experiment and the performance comparison experiment.

The experimental setup includes a suite of 360 mUCVLMP instances split up in three different size of customers: $N = \{10, 20, 30\}$, where each size of customer involves a total of 120 instances. Recently, Amazon Prime Air announced [262] that its drones can fly till $12.5km$ with a fully charged battery. This implies that current delivery drones have an area range of $491km^2$. Therefore, in order to make real instances were an autonomous truck carries multiple UAVs to perform deliveries, the total area of a problem instance needs to be higher than $491km^2$. Otherwise, the drones could fly from a single point, such as the depot, covering all the deliveries, and no UGV would be required. In this way, we set up a fixed area size of $A = 600km^2$ for every problem instance. Also, the random map generator used is the same as the one used for the TERRA's assessment (see Appendix A). We fixed the number of clusters δ as the half of the size of customers for each different size. Then, for each size of customers, we created four different UAV configurations for the UAV's endurance and UAV's speed, following the same pattern of configurations created by Murray and Raj [179]: (i) Low-Range & Low-Speed (LRLS) , (ii) High-Range & Low-Speed (HRLS), (iii) High-Range & High-Speed (HRHS) and (iv) Low-Range & High-Speed (LRHS). On the one hand, Table 4.1 shows the speed parameters in m/s of the different configurations: the take off speed, cruising speed and landing speed; and the altitude at which the UAV will fly.

Table 4.1: Speed parameters for the UAV's configurations: Low-Speed (LS) and High-Speed (HS), and the altitude at which the UAVs will fly.

	Take off (m/s)	Cruising (m/s)	Landing (m/s)	Altitude (m)
Low-Speed (LS)	7.8	15.6	3.9	50
High-Speed (HS)	15.6	31.3	7.8	50

On the other hand, Table 4.2 shows the range parameters that define the UAV's endurance for each configuration. These parameters are taken from the work of Murray and Raj [179]. They are used by the mFSTSP to model a complex UAV's energy function that computes the UAV's endurance on each delivery. This function considers the flying stages of the mFSTSP with and without the parcel weights. That is, α^t is the consumption ratio taking off with a parcel, α^c is the consumption ratio cruising with a parcel, α^l is the consumption ratio landing with a parcel, β^t is the consumption ratio taking off without a parcel, β^c is the consumption ratio cruising without a parcel, β^l is the consumption ratio landing without a parcel, γ is the consumption ratio serving a customer and, δ is the consumption ratio waiting for landing. Instead, COURIER does not consider the weight of parcels on its energy function, so it only uses these parameters to compute the UAV's endurance in time terms T_{trip}, and in distance terms R, as defined at the beginning of section 4.1 (see Figure 4.1). Appendix C shows how these values are computed specifically. In this way, we set up a fixed parcel weight for every instance of $1kg$ in order to perform a fair comparison.

Table 4.2: Power consumption parameters for the UAV's endurance: Low-Range (LR) and High-Range (HR), taken from the work of Murray and Raj [179].

	α^t	α^c	α^l	β^t	β^c	β^l	γ	δ
		$(W/(kg \cdot \frac{m}{s}))$			$(W/\frac{m}{s})$		(W)	(W)
Low-Range (LR)	11	5.5	11	22	11	22	200	400
High-Range (HR)	11	5.5	11	24	12	24	225	450

Then, the battery capacities of the UAVs for the different configurations are shown in Table 4.3. These capacities keep the same values as the ones evaluated in Murray and Raj [179]. Also, note that the highest value of $500kJ$ is computed to represent the real range $R = 12.5km$ of the Amazon's Prime Air drone commented above.

Table 4.3: Battery capacity for the four UAV's configurations: LRLS, LRHS, HRLS, HRHS.

Battery Capacity (kJ)	High-Speed (HS)	Low-Speed (LS)
High-Range (HR)	500	400
Low-Range (LR)	250	200

Each UAV's configuration was tested on 30 instances split up in three configurations with different number of UAVs: $N_{uavs} = \{1, 5, 10\}$. Thus, each configuration of available UAVs executes 10 different instances for a single UAV configuration in a particular size of customers instance. Then, the total number of instances can be expressed as: $N_{instances} = 3 \cdot 4 \cdot 3 \cdot 10 = 360$ instances.

In addition to the above parameters, every generated instance has a set of fixed parameters in order to execute either COURIER or mFSTSP: the depot location $(x = 0.5, y = 0.5)$ in the $\mathbb{R}^2$ Euclidean space, the UGV's speed $V_{ugv} = 11m/s$ and the UAV service time $Td = 60s$ shared by both models; and the UAV launch time $s^L_{v,i} = 60$, the UAV recovery time $s^R_{v,k} = 30$ and the maximum payload capacity $2.3kg$ only for the mFSTSP model. These last three values were extracted from the numerical analysis in Murray and Raj [179].

Finally, aiming to enable future comparisons with other algorithms in the literature, we present in Appendix C an extension for the standard format TSPLib [252]. This extension has been used to create every instance during the experimentation process, and it contains all the above explained parameters to be executed by any algorithm solving the last-mile delivery problem.

4.3.2 *Characterizing the COURIER algorithm*

The objective of this experiment is to assess the COURIER algorithm by analysing its behaviour under different UAV's endurance parameters and different number of UAVs, for configurations with different number of parcels. For that, we executed COURIER over the 360 generated instances and gathered the results shown in Figure 4.20. In the following, we provide an analysis of the impact for different UAV's configurations, the impact of having more or less UAVs available, and its scalability.

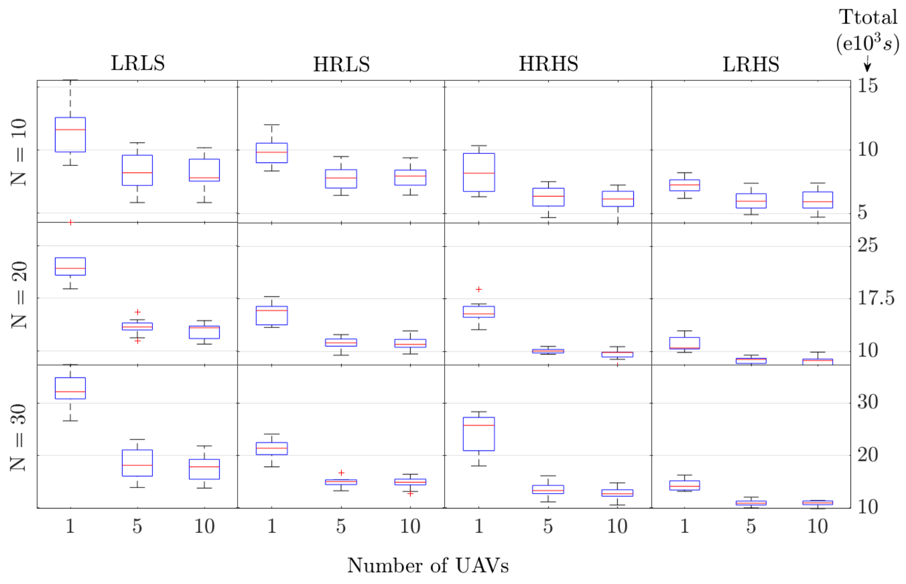


Figure 4.20: Results of the COURIER execution in the 360 generated instances. These instances represent four UAV's configurations (LRLS,LRHS,HRLS,HRHS), three numbers of parcels ($N = \{10, 20, 30\}$), and three numbers of UAVs available ($N_{uavs} = \{1, 5, 10\}$). The results show the total delivery time T_{total}. Each row of number of customers (N) is depicted in a different scale to make it easier the comparison among the UAV's configurations. Also, in spite of the different scales, it can be noted the difference in magnitude among the number of customers configurations.

From the analysis of deploying different UAV's configurations, we can firstly highlight that the most significant parameter is the UAV's speed. Figure 4.20 shows, from left to right, the best UAV's configurations, and, as can be noted, all the customer size configurations follow the same order: (4) LRLS, (3) HRLS, (2) HRHS, (1) LRHS. That is, the two best configurations are HRHS,LRHS, and the two of them have high speed parameters for the UAVs. That is, higher speed values enable the UAVs to travel faster over the same area, which minimizes the delivery times as expected. Also, these higher speed values imply the highest battery levels of the configurations in Table 4.3, which result in larger UAVs coverage area. Hence, this produce a shorter UGV's path because of the Voronoi algorithm at the first stage (see the Voronoi search method in chapter 3). Then, as the UGV is slower than the UAVs, minimizing the UGV's travel time results in being more significant than minimizing the UAV's travel time, and thus, configurations with higher speed values complete the deliveries in the shortest time.

Secondly, it is interesting how the LRHS configuration presents better results than the HRHS, which might contradict the previous affirmation about the higher battery levels are desirable to lower battery levels. Nevertheless, the combination of low UAVs' endurances and high UAVs' speed produce scenarios where the parallel deliveries experience an increment among the UAVs. It is true that this combination makes the UGV to increase its UGV's travel time because of the Voronoi algorithm. However, when combined with a high speed of the UAVs, it opens several ways to the algorithm to find complex entanglements (as explained in subsection 4.2.1). This effect produces the increment of parallel working, and so, the decrement of the time when the system is not performing a delivery. Note how the functions F_{tc} and F_{free}, explained in subsection 4.2.2, aim to minimize the time when the UAVs are not delivering, and so to maximize parallel working. Then, we can state that the maximization of the parallel working is more significant than the minimization of the UGV's travel time. From this analysis, we can assert the following Lemma 4.3.1:

Lemma 4.3.1. There is not need for deploying UAVs with huge and expensive batteries that cover a a large area. It is preferable to deploy faster UAVs with low battery capacities, than faster UAVs with too high battery capacities.

From the analysis of the number of UAVs available, we want to assess the polynomial performance evolution with the different UAVs available, and so, to acknowledge how many UAVs are enough to minimize the delivery time on these instances. From a business perspective, this is an important question in a logistic company, because reducing the operational costs of the UAVs and associated infrastructure allows to maximize the profit.

As a first observation, we can see relevant differences between solving an instance with 1 UAV and 5 or 10 UAVs. In fact, we see that this difference becomes larger as the number of parcels increases. The increment of the number of parcels along with the increment of the number of UAVs enable the algorithm an explosion of combinations to increase the parallel working in the deliveries. So, the greater the parallel working the minimum is the total delivery time, and the maximum is the difference between having 1 UAV and 5 or 10 UAVs.

A second observation is related to the difference between solving the instances with 5 and 10 UAVs. We can observe that the results are almost the same between these two configurations in every plot. This is because COURIER splits the task planning problem into sub-tours, so even for $N = 30$ instances, the number of parcels per sub-tour is not high enough in comparison to the number of UAVs. Thus, there is not much difference between doing a sub-tour of 7 parcels with 5 UAVs (of 5 UAVs available in total) than doing this sub-tour with 7 UAVs (of 10 UAVs available in total). The increment of parallel working is minimum, and so, the difference between configurations is minimum as well. Also, the set of instances with $N = 30$ presents a slightly greater difference between these configurations. This is due to the increment of available UAVs, which makes the algorithm alternate them as explains the F_{tc} function in subsection 4.2.2, and so it reduces the divorce delays. Nevertheless, this slight improvement is not worth the cost of deploying more UAVs. Thus, we can hypothesize that at some particular number of UAVs, the algorithm starts to decrease its performance for a particular set of instances, and also, this impact is heavier for instances with more number of parcels.

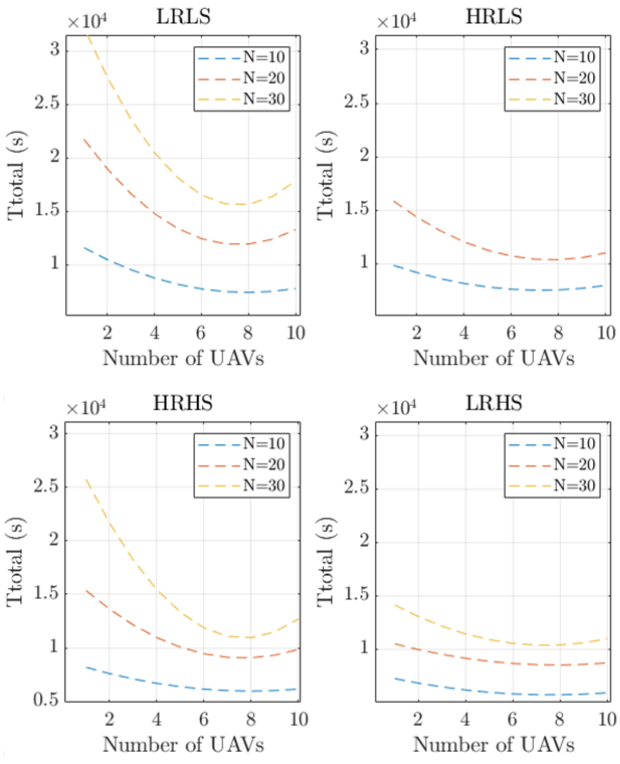


Figure 4.21: Impact of the number of UAVs available in the different configurations for the 360 generated instances. Each plot shows a polynomial regression of the total delivery time T_{total} as a function of the number of UAVs. The HRLS plot shows that $N = 30$ is equal to $N = 20$.

Aiming to demonstrate this hypothesis, we show the Figure 4.21, which shows the impact of number of UAVs in the total delivery time in the different configurations and for all the generated instances. At first, we observe the different impacts experienced by the configurations with different number of parcels. The mentioned slightly greater difference between $N = 30$ and the other configurations is now clearly identified. That is, we can observe that the increment of the number of UAVs in $N = 30$ configurations produces an almost exponential decrement in the total delivery time. Also, this decrement flattens out as the number of parcels decrease, because of the first observation, i. e., less parcels require less UAVs. Also, we can observe that each plot estimates the same global optima in the number of UAVs required to optimize the performance, i. e., $N_{uavs} \simeq 7$. Therefore, the hypothesis is demonstrated to be true, and we can assert the following Lemma 4.3.2:

Lemma 4.3.2. The polynomial increment of the number of parcels demands a linear increment of the UAVs available.

4.3.3 *Performance evaluation compared to the mFSTSP heuristic approach*

The objective of this experiment is to assess the performance of COURIER taking the heuristic solution for the mFSTSP [179] as the baseline[5]. For that, we executed COURIER and mFSTSP-h over the 180 instances related to the configurations with higher speed (HRHS and LRHS), because of their promising results in the previous characterization analysis. In the following, we first provide a theoretical comparison between the cooperation problems i. e., the mUCVLMP and mFSTSP, to highlight their relevant features. Second, we conduct a performance evaluation in the total delivery times, and thirdly, we provide an analysis between the UGV's travel time and the total UAVs' travel time.

As we already mentioned in subsection 4.3.1, in spite of both algorithms (COURIER and mFSTSP-h) exploit different cooperation approaches, both solve problems (mUCVLMP and mFSTSP) that are framed as generalizations of the last-mile delivery problem. This is the essential common feature that allows the performance evaluation gathered in this section. Nevertheless, before beginning the comparison, we introduce the key features of both approaches to set up a proper reference frame where to carry out the comparison. In this way, Table 4.4 shows these key features for both problems.

The *UGV+UAVs cooperation* and *Heterogeneous UAVs* features note that both deploy a heterogeneous multiple UGV-UAVs system to solve the problems. The *Rendezvous delay* points out that they have different cooperation models, which involves different times to compute the

5 From now on, we refer to the heuristic algorithm solving the mFSTSP, as the mFSTSP-h. Please note that both were implemented by Murray and Raj [179]

Table 4.4: Relevant features of the cooperation approaches deployed by the COURIER algorithm to solve the mUCVLMP, and the heuristic solution to solve the mFSTSP.

Features	COURIER	mFSTSP-h
UGV+UAVs Cooperation	✓	✓
Heterogeneous UAVs	✓	✓
Rendezvous delay	Tw (UGV)	UGV + UAVs
Who carries the deliveries?	UAVs	UGV+UAVs
Parallel deliveries	✓	✓
Objective function	min Ttotal	min Ttotal
UAV's energy function	Simple	Linear function
Search space	Unknown nodes	Fixed nodes
Task planning	Battery recharging	Only actions

total delivery time. In COURIER, the flight phases of the UAVs are: (1) Take off, (2) Fly to customer, (3) Deliver, (4) Fly back to UGV, and (5) Land on the UGV. Here, the deliver phase might include any kind of delivery procedure in a constant time. Instead, the UAVs land in the customer location to perform the delivery in mFSTSP-h, which adds two more phases: land and take off in the customer location. This is because the mFSTSP-h considers the weight of the parcel. In this way, the *Rendezvous delay* describes that the UAVs can wait in the customer location for the UGV in the mFSTSP-h, but only the UGV can wait for the rendezvous in COURIER. This is because, mFSTSP-h allows both systems to carry out the deliveries, whereas COURIER only allows the UAVs.

As well, Table 4.4 shows that both algorithms allow parallel deliveries. Also, COURIER describes a simple UAV's energy function to simplify the solution, but mFSTSP-h introduces a linear function based on the parcel weight, speed and operation time. The *Search space* is maybe the most relevant feature to measure and distinguish the computational performance between them. The search space in COURIER is built up by deliveries in which each parcel has its own geometrical features, and so, different rendezvous delays percentages $Tw_{\%}$ on each gene (see Figure 4.12 in subsection 4.2.2). This means that a value of $Tw_{\%} = 50\%$ has a particular meaning for every parcel, which makes each parcel to have a whole different set of vertices to look for. This causes an explosion in the size of the search graph. Instead, the search space in mFSTSP-h is formed by a set of fixed nodes which are the depot location and the deliveries location. As can be noted, the size of the search graphs is clearly biased by this number of nodes. Lastly, the *Task planning* feature distinguishes between the two cooperation models, because COURIER computes a task planning including the

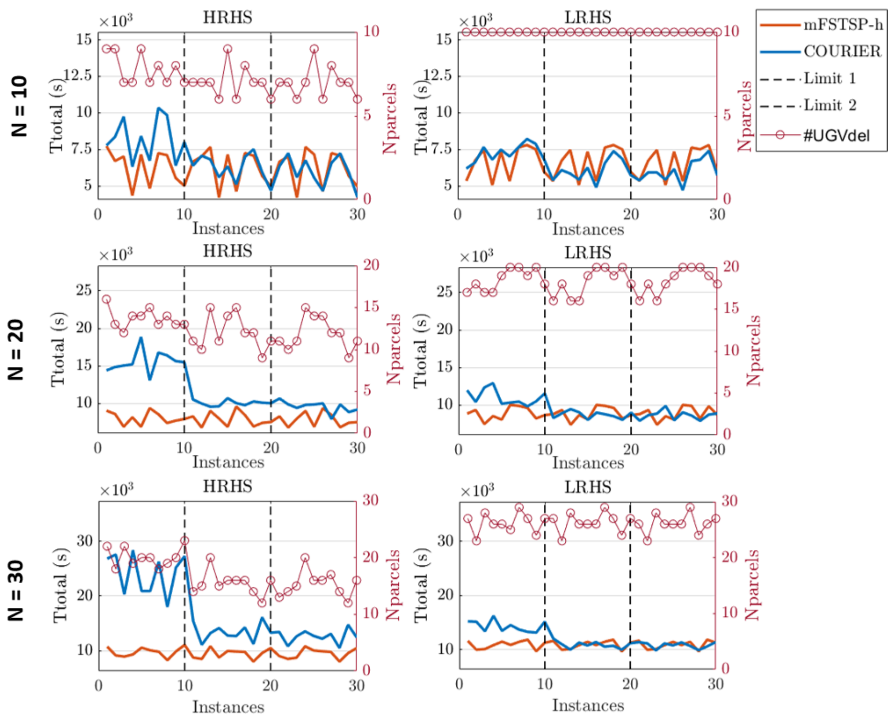


Figure 4.22: Results of the COURIER and mFSTSP-h execution in the 180 HRHS and LRHS instances, for each number of parcels $N = \{10, 20, 30\}$. Each plot shows the total delivery time T_{total} (left y-axis) of both algorithms (COURIER = blue line, mFSTSP-h = orange line) per instance (x-axis), and the number of deliveries $\#UGV_{del}$ (red right y-axis) carry out by the UGV in the mFSTSP-h. The black dashed lines denote the limits between the instances with different number of UAVs. From 1 to 10 are instances with 1 UAV, from 11 to 20 are instances with 5 UAVs, and from 21 to 30 are instances with 10 UAVs.

battery level of the UAVs on each instant. Instead, mFSTSP-h considers a full battery every time the UAV takes off to perform a delivery.

From the performance analysis, we aim to compare both algorithms to obtain their performance on the different generated instances. For that, Figure 4.22 shows the results of executing both algorithms for the HRHS and LRHS instances for the three configurations of the number of parcels $N = \{10, 20, 30\}$ and number of UAVs (black dashed lines on each plot). It specifically shows the total delivery time T_{total} in seconds for every computed instance of each algorithm. As mFSTSP-h can performs deliveries with the UGV, we also show the number of deliveries $\#UGV_{del}$ carried out by the UGV in the mFSTSP-h.

Firstly, we observe that mFSTSP-h outperforms COURIER in all the instances with 1 UAV. This is due to mFSTSP-h allows deliveries using the UGV. Thus, minimizing the number of UAVs does not impact mFSTSP-h at all. Even so, COURIER presents similar results in some instances in the LRLS configuration with $N = 10$. Also, we observe in

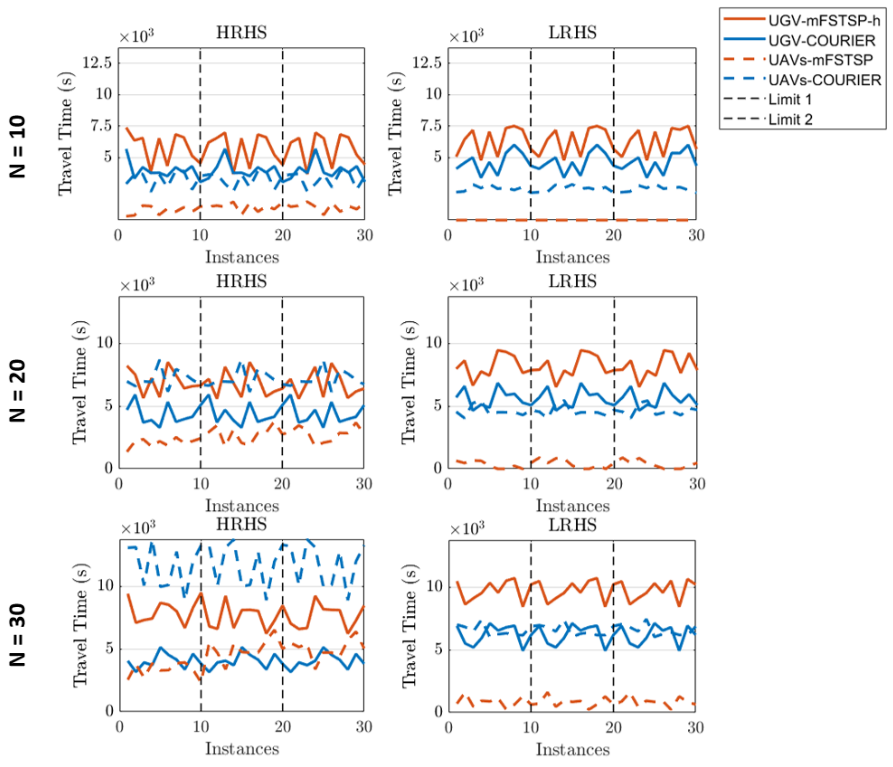


Figure 4.23: Results of the COURIER and mFSTSP-h execution in the 180 HRHS and LRHS instances, for each number of parcels $N = \{10, 20, 30\}$. Each plot shows the UGV's travel time (normal lines) and the UAVs' travel time (dashed lines) of both algorithms (COURIER = blue line, mFSTSP-h = orange line) per instance (x-axis). The vertical black dashed lines denote the limits between the instances with different number of UAVs. From 1 to 10 are instances with 1 UAV, from 11 to 20 are instances with 5 UAVs, and from 21 to 30 are instances with 10 UAVs.

HRHS that COURIER presents similar and some better results with $N = 10$. But, it scales worse than mFSTSP-h in $N = 20$ and $N = 30$ because of the high percentage of parcels delivered by the UGV, and also, the area size of the instance. That is, mFSTSP-h finds better solutions using the UGV, when introducing more deliveries in the same space. However, if the area size is larger, or the UAVs have a lower range as in LRHS, our algorithm works slightly better. Also, we observe in LRHS that in spite of the high number of UGV's deliveries, COURIER usually finds better solutions as the number of available UAVs increases, which causes an increment in the parallel working among the UAVs.

Secondly, we note that a high percentage of the parcels ($\simeq 78\%$ in UGV Del. of Table 4.5) were carried out by the UGV in the mFSTSP-h solutions. As a result, Figure 4.23 shows the UGV's travel time and the UAV's travel time in COURIER and mFSTSP-h. Here, we observe that the UGV's travel time in the mFSTSP-h is higher in all instances than in COURIER, because of that 78%. Here, we want to point out that despite mFSTSP-h provides some theoretical better solutions, it may be not efficient in some real instances, such as urban areas, where there is a heavy traffic. Thus, the UGV can be greatly affected, and so, the total delivery time. Instead, COURIER aims in minimizing the total delivery time, but also, as it is an evolution of the TERRA algorithm, it aims to minimize the UGV's and UAV's travel time at once, as we demonstrated in section 3.4. Therefore, COURIER tries to keep in balance lower values of both variables.

Table 4.5: Summary statistics of the COURIER and mFSTSP-h execution for the 180 instances generated with the HRHS and LRHS configurations. The Gap= $((C-h)/h) * 100$, where C is the objective value given by COURIER and h is the objective value given by mFSTSP-h. The percentage is based on the minimum value obtained for each configuration. Rtime$_A$ is the runtime in seconds of mFSTSP-h, and Rtime$_B$ is the runtime in seconds of COURIER.

			Gap (%)			UGV Del. (%)	Rtime$_A$ (s)	Rtime$_B$ (s)
			Avg	Min	Max	Avg	Avg	Avg
N=10	1 UAV	HRHS	44.3	0.9	71.4	78	0.07	925.9
		LRHS	12.0	-2.4	34.5	100	0.03	794.8
	5 UAVs	HRHS	0.32	-20.1	32.9	70	0.15	678.1
		LRHS	-11.2	-32.1	7.2	100	0.11	803.4
	10 UAVs	HRHS	-5.1	-38.0	32.97	70	0.25	845.7
		LRHS	-11.8	-30.3	7.3	100	0.19	911.5
N=20	1 UAV	HRHS	108.7	53.6	172.1	68.5	0.48	1824.3
		LRHS	29.1	2.9	64.9	92.5	0.28	1623.9
	5 UAVs	HRHS	31.8	6.7	55.7	60	1.23	1689.1
		LRHS	-2.3	-15.1	28.0	91	0.87	1559.7
	10 UAVs	HRHS	24.8	-7.7	45.1	59.5	2.58	1764.6
		LRHS	-3.1	-17.4	19.9	92	1.60	1594.3
N=30	1 UAV	HRHS	172.1	116.4	227.7	66.3	1.97	3851.1
		LRHS	35.2	14.9	55.8	86.5	1.12	2477.3
	5 UAVs	HRHS	49.0	29.7	82.8	50.8	5.08	3354.0
		LRHS	-0.09	-11.6	10.6	86.8	3.19	2274.6
	10 UAVs	HRHS	38.5	23.9	63.9	50.5	8.71	2754.6
		LRHS	-0.9	-12.8	12.2	86.1	5.74	2201.4

Lastly, Table 4.5 shows the summary statistics where we show the gap as the percentage difference between the solution given the mFSTSP-h and COURIER, that is, Gap= $((C-h)/h) * 100$, where C is the objective value given by COURIER and h is the objective value given by mFSTSP-h. Also, it shows the percentage of the parcels that have been delivered by the UGV in mFSTSP-h, and the runtime of both algorithms. Following the previous observations, we want to highlight the COURIER performance in solutions with 5 or 10 UAVs. Also, we can observe the high percentage of UGV deliveries, which is an average of 78%. Finally, it is relevant to assess the runtime differences between both algorithms. This is explained because of the dimensionality difference between the search spaces. As we explained with Table 4.4, the search space of COURIER is vastly larger than the search space of mFSTSP-h. Nevertheless, COURIER, as well as TERRA, are off-line task planning algorithms that compute a solution before the delivery starts, and so, the runtime is not considered as a significant constraint for these kind of off-line task planners.

4.4 SUMMARY

The last-mile delivery problem is an optimization problem that has been addressed by logistic companies to improve their efficiency in the parcel delivery and so, to increase their profit. In this way, the heterogeneous multiple UGV-UAVs system is being intensively studied since companies like Amazon or DHL announced in 2015 their interest in deploying these systems to solve the last-mile delivery. Nowadays, the literature is increasing exponentially, providing solutions with systems to minimize different objectives. In this chapter, we presented a novel formulation as a generalization of the last-mile delivery problem, called as mUCVLMP, which objective is to minimize the total delivery time. Then, we described the task planning algorithm called COURIER, which follows the first four stages of the TERRA path planner presented in chapter 3, and implements a new five stage for the task planning issue. This stage is formed by a memetic algorithm with a particular heuristic search procedure to overcome the mUCVLMP, and an arithmetic solver implementing a particular cooperation model of the heterogeneous multiple UGV-UAVs system to get the final solution. The experimental results of COURIER demonstrated that the algorithm solves the mUCVLMP as a generalization of the last-mile delivery problem. Also, they enabled us to understand its good behaviour depending on the UAVs' configuration and the number of UAVs. Additionally, COURIER shows a huge potential to be adapted for future commercial applications such as delivering supplies in war zones or catastrophic areas.

5 AN AUTONOMOUS CONTROLLER FOR COOPERATIVE MULTI-ROBOT SYSTEMS

So far, the research presented in this dissertation has been focused on cooperative planning algorithms that deliberate solutions to overcome exploration and last-mile delivery problems efficiently. However, by themselves, these algorithms cannot fully take control of the operation of robotic systems. Instead, autonomous controllers for multi-robot cooperation, such as the presented in section 2.3, are commonly used for this purpose. These autonomous controllers are software entities which integrate different modules to operate a robotic team with a certain level of autonomy. In particular, our algorithms represent just the decision-making module of these autonomous controllers.

In this chapter, we aim to contribute to the research efforts for achieving a high-level autonomy of cooperative multi-robot systems. We present Autonomous coopeRatIve Execution System (ARIES), an autonomous controller designed for cooperative multi-robot systems, compatible with any robotic domain, adaptable to any problem complexity and qualified to respond to failures. The ARIES controller is built on the Teleo-Reactive EXecutive (T-REX) architecture [263], a multi-agent architecture to control robotic systems. A brief description of T-REX is provided in the next section. Considering its advantages, our objective is to extend the T-REX architecture to build a cooperative controller following the hierarchical leader–follower paradigm. Section 5.2 describes the planning paradigm used by ARIES. Section 5.3 describes the hierarchical execution flow and the main elements of the controller. Then, we show the results of a simulation experiment performed to demonstrate the ARIES capabilities in section 5.4.

5.1 A T-REX OVERVIEW

The purpose of this section is to introduce the T-REX architecture [263, 264] to the reader. T-REX is a goal-oriented system that follows the timelines-based planning paradigm [265–267]. As Mayer et al. [268] describe, the timelines-based planning paradigm aims to control complex physical systems through the synthesis of desired temporal behaviours (called timelines) over robotic features with associated temporal functions. In timeline-based planning, these time-varying robotic features are called state variables, e. g., the navigation system. The values that a state variable can take over time are named as tokens. Then, a timeline referred to a state variable is a sequence of tokens that defines its temporal evolution, such as the navigation control of a rover reporting its current position in real time. The task of a planner is to find a temporal allocation of tokens that brings the timelines into a desired state, satisfying the rules and special conditions known as goals.

In T-REX, the state variables are distributed into modules called reactors. A reactor manipulates specific information, such as the navigation system managing just the positioning and navigation policy of a rover. State variables are exclusive property of only one reactor. There are two classes of state variables:

- *Internal* state variables are controlled by the owner reactor. The only reactor that defines a state variable as internal has sole responsibility to update it via observations. Also, goals posted by other reactors to this state variable can be considered by the reactor to plan ahead on any goal state.
- *External* state variables provide the world information to a reactor that enables to define its own internal state, but the reactor has no control over them. However, the reactor can request a specific state on an external variable posting goals to the owner reactor.

Each T-REX reactor encapsulates the planning and execution processes into a control loop modelled as a state machine. The frequency at which the state update happens is dictated by a central clock, which uses the *tick* as the time unit. The duration of a tick is a matter of design. The control loop of a T-REX reactor is based on three ordered steps:

- *Synchronization* to keep updated its internal state variables by monitoring the external state variables. That is, a reactor update may produce a goal execution over an internal state variable of another reactor.
- *Dispatching* goals among reactors in a timely manner. Each reactor defines its own dispatching window to identify when tokens (following the token start time) on an external state variable should be transformed into goals for the reactors that declares the state variable as internal. The dispatching window, denoted as H_D, is defined as in Equation 5.1, where τ is the execution frontier that expresses the current tick of the execution, λ is the latency defined as the maximum number of ticks that a reactor has to deliberate and π is the planning horizon of a reactor representing the look-ahead for deliberation.

$$H_D = [\tau + \lambda, \tau + \lambda + \pi] \tag{5.1}$$

This implies that a reactor A dispatches a goal to a reactor B (owner of the state variable) as soon as the start time of the goal intersects the H_D of the reactor B. The dispatching window guarantees that each reactor has enough time, thanks to λ, and enough information, thanks to π, to deliberate on goals provided by other reactors.

- *Deliberation* to plan a sequence of tokens aiming to achieve the received goals on every reactor.

Therefore, a T-REX agent is formed by a synchronized set of reactors, whose objective is to change the environment of the robotic platform through actions on their state variables. This division enables the problem to be distributed into several reactors in different abstraction levels. Typically, we can differentiate deliberative and executive reactors. Deliberative reactors are aware of the high-level mission goals and could require larger temporal scopes (λ) for problem solving. Instead, executive reactors are in a lower abstraction level and commonly implement reactive behaviours. Then, the T-REX engine is in charge of synchronizing all reactors, so they can be executed concurrently and all are notified of the evolution of that (external) state variable that affect them, being able to deliberate and dispatch appropriated plans.

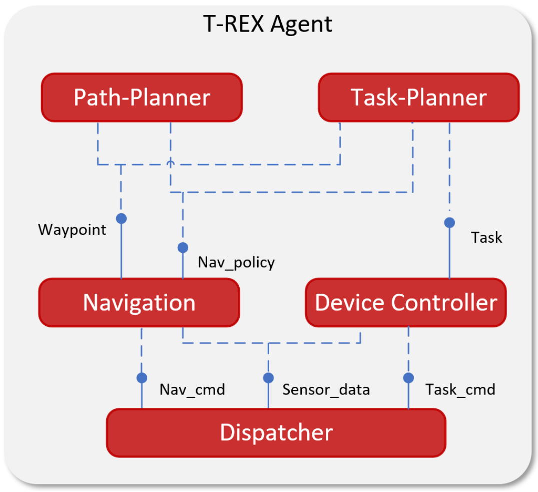


Figure 5.1: A T-REX agent is formed by multiple reactors (red boxes) which are connected through state variables provided by one reactor as internals (solid lines) and available for other reactors as externals (dashed lines).

We present an example of a T-REX agent in Figure 5.1. This T-REX agent is formed by two deliberative reactors (*Path-Planner* and *Task-Planner*), two executive reactors (*Navigation* and *Device Controller*) and the *Dispatcher* reactor. The *Path-Planner*, which has the *Waypoint* state variable as external, computes the routing for the vehicles and reports each target point to *Navigation* by posting a goal to the *Waypoint* state variable. Then, *Navigation* updates the *Waypoint* state via posting an observation when the target point is achieved. The *Dispatcher* is in charge of communicating with the robotic system. As can be observed, the communication among reactors is carried out via observations and goals to the state variables.

Therefore, T-REX is an architecture which integrates both the deliberation and reaction processes within a single agent to cover the high and low-level mission management. Nevertheless, the current T-REX version presents the following limitations in multi-robot cooperation domains: (i) centralized execution, that is, T-REX was originally built to be deployed in a centralized system which can execute a single or multiple T-REX agents and; (ii) centralized communication, which means that T-REX does not support the communication among distributed agents executed in different robotic systems. These limitations make a tough task to bring up a T-REX based architecture which could deal with any multi-robot cooperation scenario.

The autonomous controller presented in section 5.3 represents a hierarchical architecture to enhance T-REX for its deployment in any cooperative multi-robot domain. To do that, we have integrated the hybrid leader-follower paradigm (discussed in subsection 2.3.4) into T-REX. This hybrid paradigm can upgrade the T-REX based architecture in the following points: (i) distributed execution, that is, capabilities to response to individual robot failures without leading to a complete team failure and to be scalable to the problem complexity and; (ii) hierarchical deliberation, which centralizes the high-level planning into the leader, gives the capability to coordinate changes on the agent's behaviours in response to a dynamic environment or changes in the team.

5.2 FROM TEMPORAL ACTION-BASED TO TIMELINE-BASED PLANNING

As we have specified in the section above, the T-REX architecture follows the timeline-based paradigm to model the world of the robotic system. Recent studies about the timeline-based planning [269, 270] demonstrate that it has not been reached a complete characterization of this paradigm complexity to model the behaviour of robotic systems. For now, these studies demonstrate that simplified formulations of timeline-based problems are EXPSPACE-complete and then, can be expressive enough to be stated as action-based temporal planning problems. Nonetheless, it is still unclear where the complexity border between the timeline-based planning and the action-based temporal planning is.

In this way, we have chosen the action-based temporal planning for the following two reasons: first, action-based is commonly used in the planning community, so there is a broad variety of action-based temporal planners available to be used in ARIES; and second, to open up an initial step to encode a temporal action-based paradigm such as Planning Domain Definition Language (PDDL) [271] into a timeline-based paradigm. Bernardini and Smith [272] present a complete translation from a temporal action-based planning, such as PDDL2.2 [273], to a timeline-based planning, such as New Domain Definition Language (NDDL) [274]. In our case, we present initial translation primitives from PDDL2.1 to the timeline-based language used by T-REX, i. e., the Domain Definition Language (DDL) [275].

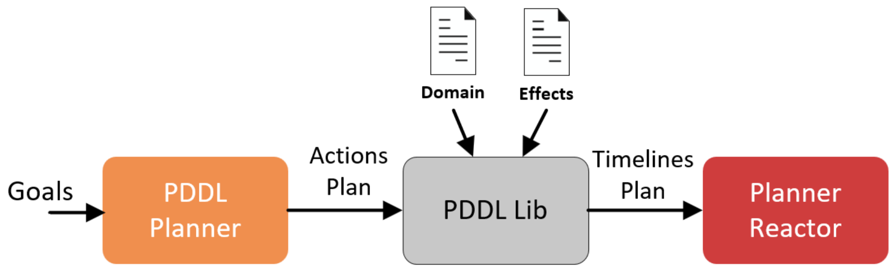


Figure 5.2: The translation flow implemented in ARIES between a PDDL Planner with action-based temporal planning, and the Planner Reactor from ARIES with timeline-based planning. *PDDL-Lib* is able to encode an actions plan as a tokens plan thanks to the domain and effects files.

Contrary to the timelines-based paradigm used by T-REX, an action-based temporal language such as PDDL2.1, uses predicates logic and the world is seen as an entity that can be in different states. The domain specifies actions that can be performed to change the world state and, only applicable when some particular states are set. This paradigm aims to find a sequence of actions that, from an initial world state, through applying successive actions, the system achieves a desired goal state. So, as T-REX and PDDL2.1 are not initially compatible, a tailored solution was required to merge the action-based paradigm used by PDDL2.1 to the timelines-based paradigm accepted by T-REX.

Therefore, we developed a common language between PDDL and T-REX that translates the sequence of actions of a PDDL-based planner into a sequence of tokens over their state variables with temporal constraints. This translation is performed thanks to the *PDDL-lib* as Figure 5.2 shows, a PDDL library developed by Muñoz et al. [276] to extract relevant information from PDDL files, execute any PDDL-based planner and read the generated plan in order to execute it. Particularly, *PDDL-lib* requires the domain file and an effects file to translate a sequence of PDDL actions (action-based temporal paradigm) as a sequence of tokens (timeline-based paradigm). In the following, we present the functional scope of both configuration files:

- The PDDL domain file, as we explained above, describes the actions required to change the world state. We use a particular naming convention which allows *PDDL-Lib* to identify PDDL actions as tokens on their specific state variables. Notwithstanding this naming, we have not reached a complete translation of the

PDDL domain file, due to semantic differences that we have not considered yet, for instance, the effects and conditions of every action (this is the reason we created the effects file explained below). Thus, we can only guarantee a translation for simplified domain formulations. Currently, we are able to translate the PDDL predicates and temporal actions as tokens belonging to state variables as follows:

- A PDDL predicate follows the format *(StateVariable_Predicate $A_1 ... A_n$)*. *PDDL-Lib* translates it as the token *Predicate* with the attributes $A_1 ... A_n$ belonging to *StateVariable*. The following Figure 5.3 shows an example of such formulation. For instance, *UGVBase* is the state variable of the predicate *At*, whose attributes are: *?i*, *?x* and *?y*. Another example, *UAVBase* is the state variable of the predicate *At*, whose attributes are: *?i*, *?x*, *?y* and *?z*.

```
(:predicates
     (UGVBase_At ?i - Id ?x - X ?y - Y)
     (UAVBase_At ?i - Id ?x - X ?y - Y ?z - Z)
)
```

Figure 5.3: An example of PDDL predicates definition following an equal action-based and timeline-based form.

- A PDDL temporal action is named as *(:durative-action StateVariable_Action)*. *PDDL-Lib* translates it as the token *Action* belonging to *StateVariable*. Figure 5.4 shows an example of this formulation. For instance, *UAVBase* is the state variable of the action *TakingOff*.

```
(:durative-action UAVBase_TakingOff
  :parameters (?i - Id ?x - X ?y - Y ?z - Z)
  :duration {…}
  :condition {…}
  :effect {…}
)
```

Figure 5.4: An example of a PDDL temporal actions definition following an equal action-based and timeline-based form.

- The effects file allows *PDDL-Lib* to understand the actions plan generated by the PDDL Planner, and to build a timelines plan following the nomenclature of the domain file. As we do not guarantee a full translation from the PDDL domain file, we create the effects file to define an effect token for every temporal action defined in the PDDL domain file. That is, this file declares the effect and parameters of every temporal action as a statement with the format *(StateVariable_Action M StateVariable_Predicate)*, where *Predicate* is the effect of *Action* and *M* is the number of parameters that *PDDL-Lib* has to assign to *StateVariable_Predicate* from *StateVariable_Action*. Figure 5.5 shows an effects file example, where *UAVBase_At* is the effect of *UAVBase_TakingOff*, and *PDDL-Lib* captures the last four parameters of this last predicate: *?i*, *?x*, *?y* and *?z* shown in Figure 5.4. Another example, *UGVBase_At* is the effect of *UGVBase_GoingTo*, and it gets the last two parameters *?x* and *?y*, which represents the destination location.

```
(UAVBase_TakingOff 4 UAVBase_At)
(UGVBase_GoingTo 2 UGVBase_At)
```

Figure 5.5: An example of an effects file for a heterogeneous simple UGV-UAV system example.

Once the sequence of tokens has been built for every state variable, *PDDL-Lib* needs to get the duration of every token to complete the timeline of every state variable. In this way, *PDDL-Lib* captures this duration from the plan computed by the PDDL Planner.

5.3 THE ARIES AUTONOMOUS CONTROLLER

The objective of ARIES [277] is to provide an adjustable controller for deploying cooperative heterogeneous/homogeneous robot teams. In this way, we have implemented a multi-agent architecture based on the T-REX system and supported on standardized technologies for enabling compatibility with other robotic systems (flexible), adaptability to different problem domains (scalable) and to be qualified to respond to some robot failures (fault tolerant).

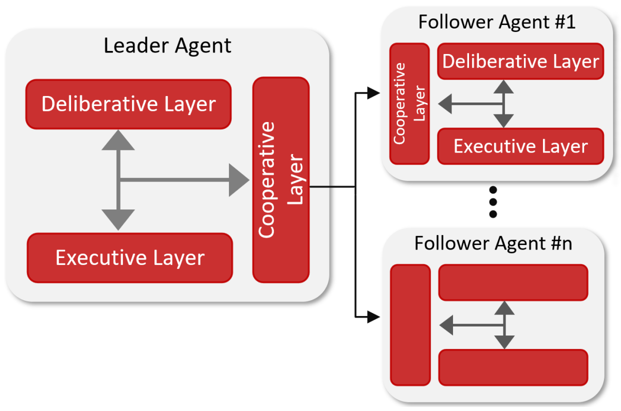


Figure 5.6: The ARIES architecture follows a hierarchical scheme using the hybrid leader-follower approach, where there is a leader agent and one or more follower agents.

ARIES follows a hierarchical scheme using the hybrid leader-follower approach in which the leader centralizes the cooperative deliberation process of the whole robot team, and then, it coordinates the execution of each follower. The ARIES architectural concept is illustrated in Figure 5.6. The follower(s) execute(s) the goals received from the leader and report the results to it, as well as monitor(s) and repair(s) (if needed) their status in order to be fully operative for the team.

5.3.1 *The hierarchical execution flow*

ARIES represents the effort of bringing T-REX and the leader-follower scheme together by splitting the control architecture in hierarchical agents, where each T-REX agent is executed on its own robotic platform. This effort results in two types of T-REX agents: the leader agent and the follower agent. Following the T-REX design criteria, both leader and follower agents have been built under hierarchical layers.

At the leader agent's top-level, a *deliberative layer* provides the cooperative planning capabilities. At the follower agent's top-level, a *deliberative layer* provides planning capabilities for supporting fault tolerant behaviours, expressed as series of rules that trigger reactive behaviours to overcome non-nominal situations.

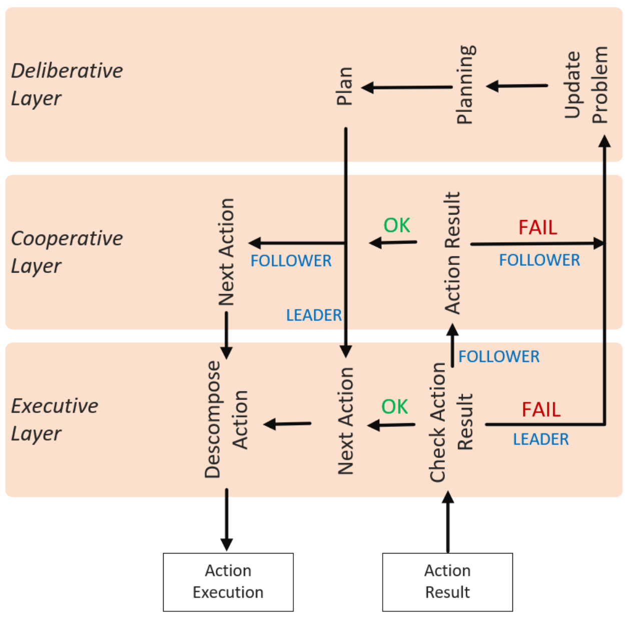
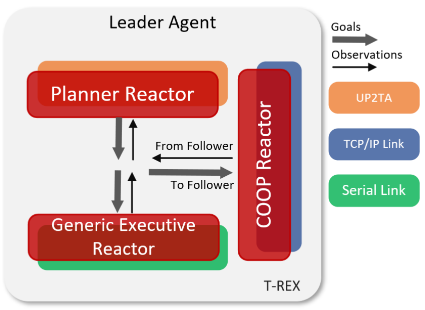


Figure 5.7: Basic execution flow in ARIES. Both leader and follower agents have been built under three hierarchical layers: the deliberative, the cooperative and the executive layer.

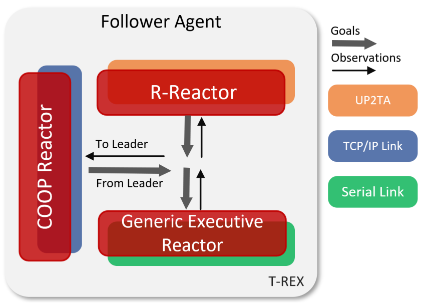
At middle and bottom levels, both agents have a *cooperative layer* for synchronizing tasks execution and an *executive layer* for execution control and monitoring. Therefore, for a given plan provided by the *deliberative layer*, the leader agent has to publish the mission goals to the *executive layer*. If the target is a follower agent, the *cooperative layer* forwards the goal to the *executive layer* of the corresponding follower. Then, the *executive layer* is in charge of the action execution and monitoring. This is the execution cycle followed in ARIES as shown in Figure 5.7. In this description, we do not have included the functional layer used to control the robotic platform. Currently, the functional support is directly integrated in the T-REX agents. However, it is possible to easily extend the T-REX agents to use different technologies for the functional layer such as GenOM [278] or ROS [279] for instance.

Regarding the fault tolerant system, ARIES is able to control specific non-nominal states. On one hand, the leader agent can manage unachieved goal(s) and communication failures. For unachieved goals, the leader follows a two steps procedure: first, the *executive layer* isolates the event and tries to execute a reactive rule that may solve the issue; if the first step fails, the second step followed by the *executive layer* is to forward the issue to the *deliberative layer*. This last implies a replanning and tasks reassignment to the followers. In such case, the followers replace their previous plan with the new plan given by the leader. For the communication failure, the leader waits for any heartbeat signal from any follower. If none follower sends a heartbeat signal, the leader performs replanning and tries to achieve the mission goal on its own.

On the other hand, the follower agent can manage the unachieved goal and communication failure in a different way. For the unachieved goal state, the *cooperative layer* of the follower reports the error to the *deliberative layer* of the leader, asking for a new plan considering the follower(s) current states. For the communication failure, there is a contingency plan when the leader is missing. In this situation, the followers enable an operational mode to communicate with its nearest follower partner, and so, to randomly select a new leader. This is an initial approach and it is not a complete failure management. Building a complete failure management system is an on-going research work.



(a) The Leader Agent in ARIES.



(b) The Follower Agent in ARIES.

Figure 5.8: T-REX agents in ARIES following the leader-follower approach.

5.3.2 *The leader agent*

The leader agent is the T-REX agent that provides the cooperative planning capabilities to an ARIES instance. It is made up of three

T-REX reactors, each one placed in a different layer as shown in Figure 5.8a: the Planner Reactor in the *deliberative layer*, the COOP Reactor in the *cooperative layer* and the Generic Executive Reactor (GER) [280] in the *executive layer*. In the following, we describe the functional scope of each layer.

At the top level, the *deliberative layer* is formed by the Planner Reactor, a PDDL-based reactor which uses two input files to represent the world knowledge: the domain and the problem file (see Figure 5.9). The domain file contains the description of the possible state variables that can be achieved by every robotic system in the team as shown in section 5.2. The interaction among the state variables of different robotic systems into the domain file defines the cooperation model of the robot team. The problem file includes the initial facts defining the starting tokens of every state variable for each robotic system in the team and, also, the high-level goals we want to achieve. This information is used by a PDDL planner to find a sequence of tokens which allows the cooperative robot team to reach the goal states from their initial states. Also, the Planner Reactor declares every state variable of the leader and every follower as external. The externals of the leader are used by the Planner Reactor to publish goals to the GER (B.1 in Figure 5.9) as we explained in section 5.1.

We have integrated the Unified Path Planning and Task Planning Architecture (UP2TA) [281] as the PDDL planner of ARIES. UP2TA is a planner that interleaves path-planning and task-planning for mobile robotics applications. Nevertheless, ARIES can be easily adapted to different PDDL planners, such as Fast Forward [282] or Subgoal Partitioning Planning [283], thanks to the use of an standardized planning language and the support provided by the *PDDL-lib*.

The Planner Reactor, as a deliberative reactor explained in section 5.1, has a certain latency $\lambda > 0$ and look-ahead $\pi > 1$ within the timing model of the T-REX engine. These values are a matter of design of the user. Then, attending to the T-REX notation, its dispatching window H_D can be stated as the Equation 5.1.

At the middle level, the *cooperative layer* is formed by the COOP Reactor, a T-REX reactor whose objective is to carry out the cooperation with the followers. Figure 5.9 shows that the COOP Reactor deploys a TCP/IP server to connect with every follower and it declares every state variable of every follower as internal. Then, when a new sequence of tokens has been planned, the Planner Reactor sends the follower's goals to the COOP Reactor (A.1 in Figure 5.9). The COOP Reactor forwards the goals to the COOP Reactor of the corresponding follower (A.2 in Figure 5.9) for execution and, so, it waits (*Leader Waiting*) for the reception of the follower observation (A.5). Then, it reports this observation to the Planner Reactor (A.6). This layer manages the communication failure, where the leader waits for any heartbeat signal from any follower. If a time threshold (defined by the user) is reached

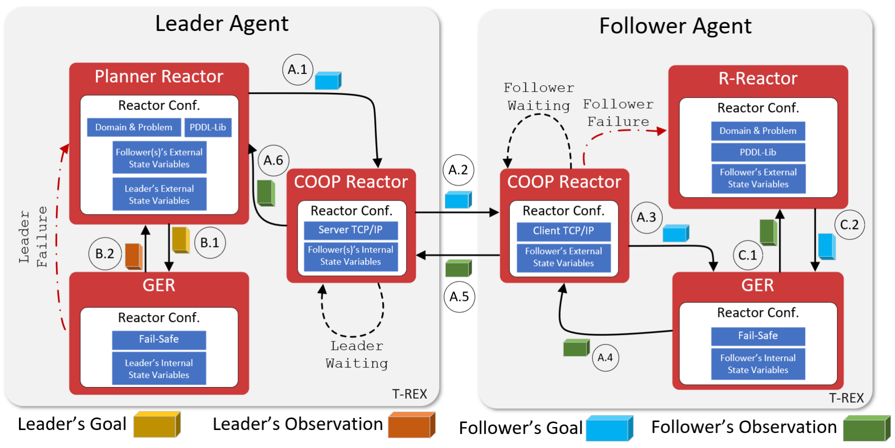


Figure 5.9: Mandatory configurations and basic work flows to deploy the cooperation in ARIES. The cooperation among two T-REX agents is carried out via observations and goals. The normal black lines represent the communication flow of observations and goals. The dashed black lines represent waiting states in the coordination among different agents. The red dashed lines represent different failure states that a reactor can communicate to other reactor.

and none follower had sent the heartbeat signal, this layer reports the wrong state to the Planner Reactor to perform replanning.

At the bottom level, the *executive layer* is formed by the GER, an execution module designed to be adaptable to any robotic domain. Figure 5.9 shows that the GER declares every state variable of the leader agent as internal. Its role is to dispatch the tokens, of its internal state variables, to the functional layer of the robotic platform. Then, when the goal has been reached, it sends the observation to the Planner Reactor (B.2 in Figure 5.9). It provides an interface between the Planner Reactor and the functional layer without creating knowledge dependencies between them. Also, it has the capability to monitor the tokens execution. This layer manages the unachieved goal failure, in which the GER sets a fail-safe mode. In this mode, it tries to repair the error as soon as possible by isolating the event from the whole system and triggering a reactive rule to solve it in a short time. The fail-safe mode requires a configuration file where the user has to define the reactive rule to trigger for every unachieved goal. For instance, if the leader's camera cannot take a picture, the user can set a reactive rule to reboot the camera that might solve the issue. Thus, it avoids to report to the Planner Reactor the failure avoiding collateral effects. However, if the failure persists after executing all possible recovering rules (*Leader Failure* in Figure 5.9), the GER notifies to the Planner Reactor, asking for replanning that may affect the whole mission.

The GER is considered as a reactive reactor, where the latency $\lambda = 0$ and the look-ahead $\pi = 1$ within the timing model of the T-REX engine. Then, its dispatching window H_D can be defined as follows:

$$\begin{aligned} H_D &= [\tau + \lambda,\ \tau + \lambda + \pi] = \\ &= [\tau,\ \tau + 1] \rightarrow \lambda = 0,\ \pi = 1 \end{aligned} \tag{5.2}$$

5.3.3 *The follower agent*

The follower agent is a T-REX agent which provides a wide range of functional capabilities in order to accomplish cooperative missions. These capabilities depend on the deployed robotic system. It is made up of three T-REX reactors, each one placed in a different layer, as shows Figure 5.8b: the Recover-Reactor (R-Reactor) in the *deliberative layer*, the COOP Reactor in the *cooperative layer* and the GER in the *executive layer*. As we did with the leader agent, the functional scope of each layer is depicted in the following.

At the top level, the *deliberative layer* is formed by the R-Reactor, a T-REX reactor focused on managing the communication failures. R-Reactor is called only when the communication between the leader and follower agent is lost (*Follower Failure* in Figure 5.9). As well as the Planner Reactor of the Leader Agent, R-Reactor uses a PDDL domain file to model behaviours for recovering the robotic system from failures, e. g., communication problems. Also, it uses the *PDDL-lib* to translate the sequence of actions into tokens over state variables with temporal constraints ready for execution. The R-Reactor declares every state variable of the follower agent as external, in order to be aware of opportunistic system failures and to act accordingly (C.2 in Figure 5.9) when a system recovery is required.

The PDDL planner used by the R-Reactor is UP2TA. The domain file contains the description of every possible failure and the sequence of tokens required to recover from them. The problem file includes the initial facts defining the initial state of the robotic system and it does not contain goals because its initial state does not have failures. A goal into the problem file means that the robotic system is stuck in a failure state and it requires the planning of a sequence of tokens to deal with it. Thus, it is a deliberative reactor, so it may require a certain latency $\lambda > 0$ and look-ahead $\pi > 1$ for the deliberation process. These values are a matter of design and its dispatching window H_D can be defined as shows Equation 5.1.

At the middle level, the *cooperative layer* is formed by the COOP Reactor, whose objective is to enable the coordination with the leader agent. It deploys a TCP/IP client with a unique identifier into the leader agent and, it declares the state variables of the follower as externals. The COOP Reactor receives the goals from the leader agent (A.2 in Figure 5.9) and, it forwards them to the GER (A.3) for their execution. Then, it waits (*Follower Waiting*) until a goal has been executed properly in the follower agent. So, when the COOP Reactor has received the observation (A.4) from the GER, it forwards the observation through the TCP/IP link (A.5) to the leader agent. If the communication with the leader agent is lost, it reports the failure to the R-Reactor (*Follower Failure*) for triggering the communication failure recovery.

At the bottom level, the *executive layer* is formed by the GER, implemented in the same way as the leader agent but with different settings, i. e., it declares, as shows Figure 5.9, the state variables of the follower agent as internals. Its role is to dispatch the tokens over its internal state variables, as goals requested by the COOP Reactor or the R-Reactor, to the functional layer of the robotic platform. Also, it has the capability to monitor the tokens execution over its internal state variables. If the unachieved goal failure arises, the GER sends every failure to the COOP Reactor to forward them to the leader agent (as in A.4, A.5 and A.6 in Figure 5.9). Then, the follower waits for a new plan from the leader.

5.4 EXPERIMENTAL DEMONSTRATION IN THE V-REP SIMULATOR

In this section we present the ARIES demonstration for a simulated study case of the ECU-CSURP discussed in chapter 3. Specifically, the study case is framed on the ECU-CSURP extension for $\mathbb{R}^3$ Euclidean spaces presented in subsection 3.3.1 and, a heterogeneous simple UGV-UAV system. Nevertheless, instead of executing TERRA, we integrated the UP2TA planner because of the following two reasons: UP2TA interleaves path planning and task planning, whereas TERRA is only a path planner; and UP2TA is a PDDL planner, which allows us to design different exploration paradigms for comparing, such as the one integrated in the LARES system [284, 285]. The LARES system represents the initial steps for deploying ARIES in a real application. For the demonstration, we tested ARIES over a nominal scenario (i. e., there are no anomalous events during the mission) simulated in the Virtual Robot Experimentation Platform (V-REP) framework [286]. Also, we designed a particular virtual reality application [287] for the assessment of the UP2TA planner and the generated mission plan. This application provides a three-dimensional view of the path generated in real Mars surfaces. Furthermore, we present some computational results of the ARIES execution in several scenarios arisen from this study case, considering different complexity levels in terms of tasks number and energy capacity.

5.4.1 *Study case. Towards a future Mars exploration with a heterogeneous simple UGV-UAV system*

This study case is based on a future Mars exploration that arises from the scientific needs to reach complex targets on Mars, such as taking pictures around a cliff, a crater or a mountain ridge, which the current

UGVs cannot accomplish due to safety and operational constraints. As well as the ECU-CSURP formulates in section 3.1, the UAV is in charge of reaching these targets. However, UAVs are robotic systems with limited energy resources to complete large duration missions on its own.

This future Martian exploration follows the cooperation synergy exploited in section 3.2, where the UGV is a moving charging station carrying the UAV between charging stops, such as a cliff edge surroundings, so the UAV can take off and obtain pictures around the cliff without flying long distances. Also, the UGV provides autonomous recharging to the UAV during the charging stops. In such mission proposal, a heterogeneous simple UGV-UAV system executing ARIES would be empowered to accomplish long-term explorations with a high level of autonomy. This mission represents just one of many future Mars missions examples.

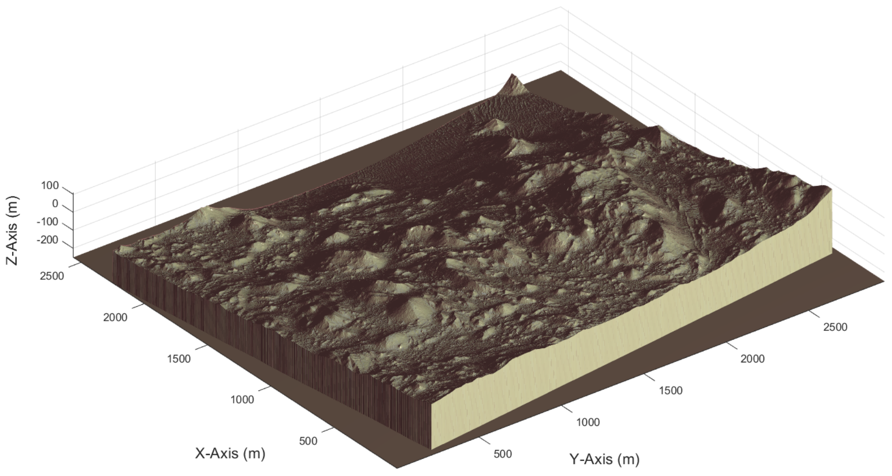


Figure 5.10: Experimental scene simulated in V-REP and used for the experimental demonstration of ARIES. The DTM if the one used for the ECU-CSURP instance presented in subsection 3.3.1.

Following the described Mars exploration, the ARIES instance deployed for this demonstration is shown in Figure 5.11. It defines the UGV as the leader agent and the UAV as the follower agent. For the UGV, we have modelled a PDDL domain and problem with the exploration mission. For the deliberative layer of the UGV, we have modelled the UGV's locomotion system through the state variable *UGVBase* and the UGV's on-board charging station as *UGVStation*. For the UAV, we have modelled the locomotion system with the *UAVBase* state variable, the zenithal camera with *UAVCamera* and the energy resource with *UAVEnergy*. The UAV energy resource allows us to model the boundaries that the UAV can reach from a take off point. Please note that this demonstration only considers a nominal scenario without anomalous events, so fail-safe configurations have not been considered.

The experimental scene was simulated in the V-REP, a flexible and scalable simulation framework. V-REP allows to use different programming techniques to implement the controllers (kinematics or dynamics, for instance) of the robotic systems. Moreover, it provides functionalities to deploy highly customizable scenes. Our experimental scene consists of a specific area extracted from a real Mars DTM as Figure 5.10 shows, and the simulated heterogeneous simple UGV-UAV system shown in Figure 5.11. Also, V-REP provides a remote interface to interact with an external entity via socket communication. We take advantage of this interface to set the communication link between ARIES and V-REP. In particular, the GER of the leader and follower agents were set up to communicate with the V-REP remote interface of each robotic system, as shows Figure 5.11.

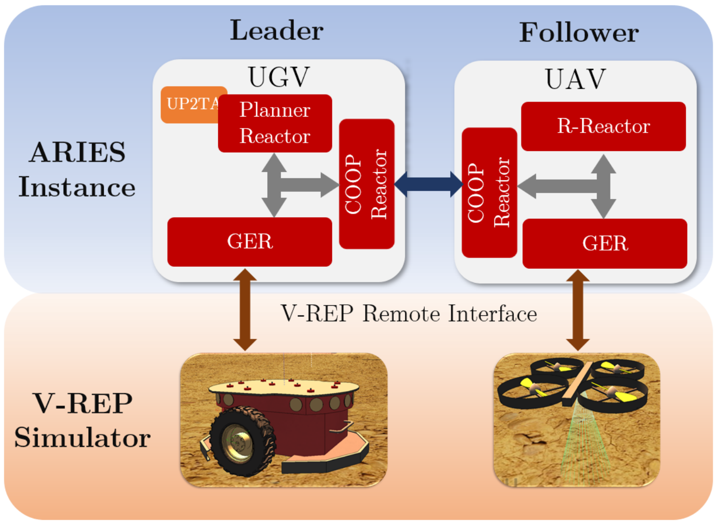
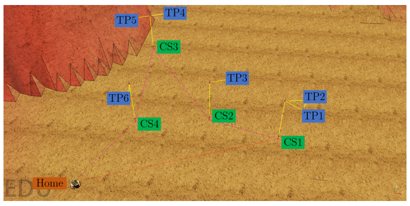


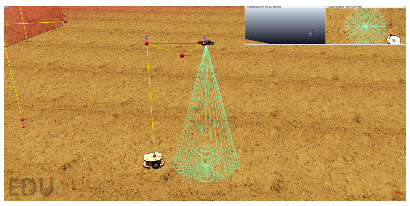
Figure 5.11: The ARIES instance deployed for the experimental demonstration in the V-REP simulator. The simulated UGV is the leader agent and the simulated UAV is the follower agent. The V-REP Remote Interface allows the communication between ARIES and V-REP.

The demonstration consists of acquiring different samples in a Mars environment. As the ECU-CSURP states, these robotics systems have been modelled as Dubins vehicles [247]. Also, we assume that the UGV does not have energy constraints, so it has enough energy resources to complete the exploration mission. Instead, the UAV energy constraint has been modelled as the maximum flight time with a high capacity battery fully charged. The nominal scenario used for the ARIES demonstration is defined as follows:

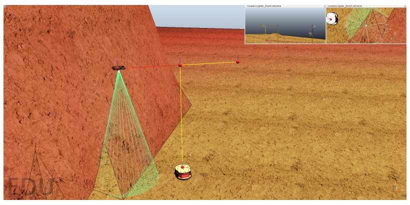
- **Nominal**. The heterogeneous simple UGV-UAV system starts at the home location with the UGV carrying the UAV, which has full battery. The objective is to take 6 pictures (TP1-6) (see Figure 5.12a). A high capacity battery allows the UAV to reach farther targets without performing too many charging stops. Figure 5.12 shows different pictures taken during the simulation.



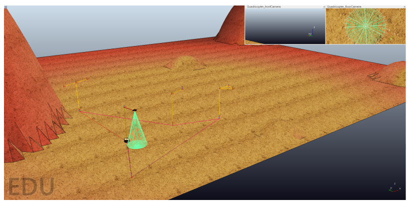
(a) Experimental scene in V-REP.



(b) Taking a picture (TP2) of the surface.



(c) Taking a picture (TP5) of a mountain hillside.



(d) Taking a picture (TP6) of the surface.

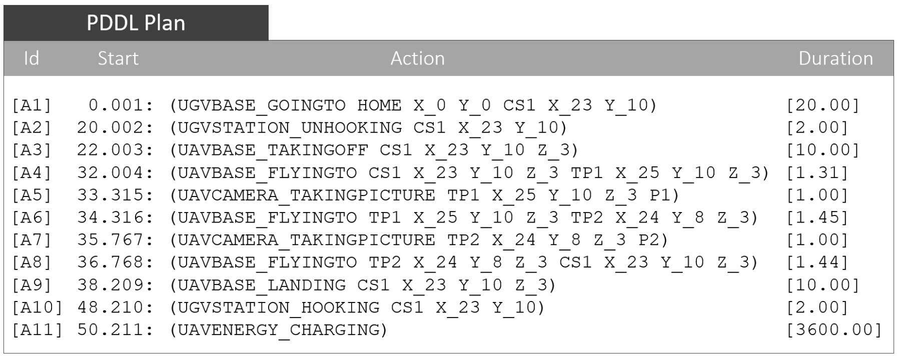
Figure 5.12: Demonstration of ARIES for the described nominal scenario. Figure 5.12a shows an instance with six target points (TP1-6) distributed around the Mars surface. The UGV carries the UAV through charging stops (CS1-4 in green color) to allow the UAV to reach the TPs (blue color) without running out of energy.

5.4.2 *Simulation results*

The simulation starts by defining the goals and the initial state of the nominal scenario, i. e., reaching the targets TP1-6 starting from the home location. Once the UP2TA planner of the leader agent receives the goals, it generates a complete plan to achieve one-by-one the targets. Figure 5.13 describes a part of the plan for reaching TP1 and TP2 from the charging stop CS1, as it is graphically represented in Figure 5.12b. Then, the generated PDDL plan is forwarded to the Planner Reactor.

The Planner Reactor is in charge of translating the planned actions into a timeline-based plan[1] as explained in section 5.2. The resulting timeline-based plan translated from the PDDL plan is shown in Figure 5.14. This timeline-based plan contains the PDDL actions (A1-11) plus the initial states (green boxes) modelled in the PDDL problem and the effect tokens (gray boxes) stated in the effects configuration file. In this way, each state variable has a complete temporal evolution from the beginning to the end of the time horizon. Now, the new plan

1 The computational cost of the translation process of the *PDDL-lib* is negligible respect to the controller execution.



PDDL Plan

Id	Start	Action	Duration
[A1]	0.001:	(UGVBASE_GOINGTO HOME X_0 Y_0 CS1 X_23 Y_10)	[20.00]
[A2]	20.002:	(UGVSTATION_UNHOOKING CS1 X_23 Y_10)	[2.00]
[A3]	22.003:	(UAVBASE_TAKINGOFF CS1 X_23 Y_10 Z_3)	[10.00]
[A4]	32.004:	(UAVBASE_FLYINGTO CS1 X_23 Y_10 Z_3 TP1 X_25 Y_10 Z_3)	[1.31]
[A5]	33.315:	(UAVCAMERA_TAKINGPICTURE TP1 X_25 Y_10 Z_3 P1)	[1.00]
[A6]	34.316:	(UAVBASE_FLYINGTO TP1 X_25 Y_10 Z_3 TP2 X_24 Y_8 Z_3)	[1.45]
[A7]	35.767:	(UAVCAMERA_TAKINGPICTURE TP2 X_24 Y_8 Z_3 P2)	[1.00]
[A8]	36.768:	(UAVBASE_FLYINGTO TP2 X_24 Y_8 Z_3 CS1 X_23 Y_10 Z_3)	[1.44]
[A9]	38.209:	(UAVBASE_LANDING CS1 X_23 Y_10 Z_3)	[10.00]
[A10]	48.210:	(UGVSTATION_HOOKING CS1 X_23 Y_10)	[2.00]
[A11]	50.211:	(UAVENERGY_CHARGING)	[3600.00]

Figure 5.13: Partial PDDL plan to reach TP1 and TP2 from CS1 in the nominal scenario. The start and duration parameters are represented in seconds. The attributes X, Y, Z follow the format *Name_Value*.

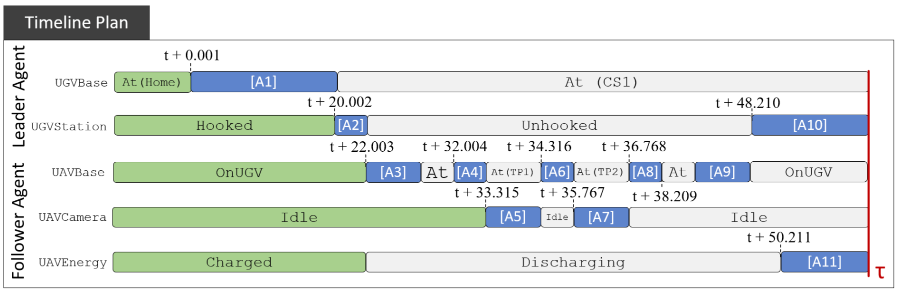


Figure 5.14: Partial timeline-based plan translated from the partial PDDL plan in Figure 5.13 for the leader and follower agents. The state variables are represented on the left side. The green tokens represent the initial state of the mission. The blue tokens represent the actions taken from the PDDL plan. The gray tokens are the effects of the tokens stored in the effects configuration file. The start of the mission is denoted by t, and the execution frontier is denoted by τ.

can be executed by an architecture which follows the timeline-based paradigm such as ARIES.

Once the plan is translated, the Planner Reactor starts the execution token by token. Figure 5.15 shows the execution flow for achieving TP1 and TP2. First, we observe that every planned token has its own effect token, e. g., the *TakingPicture* token has the *Idle* effect. Thus, every token execution in a state variable implies an effect, which means that ARIES requires two tokens exchange to carry out the coordination between the agents. Second, we can observe that both agents represent the *UAVCamera* state variable evolution, which means that both agents share information in order to accomplish a proper coordination. This information sharing enables the design of failure recovery models for future scenarios where both agents coordinate a joint plan to solve anomalous situations. Finally, we observe how the dispatching process has been split by a coordination process in which both agents communicate to hold a truthful coordination.

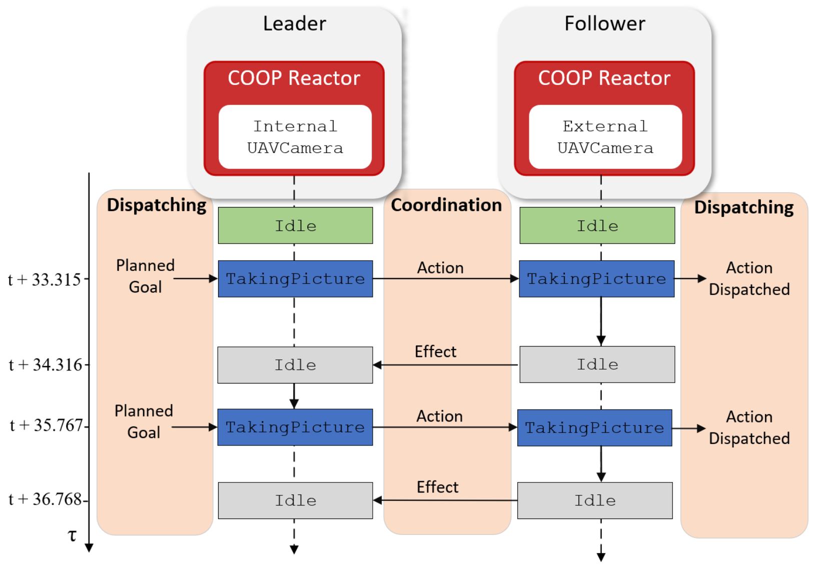


Figure 5.15: Execution flow for taking the pictures TP1 and TP2 whereas the UGV is at CS1. Green tokens represent the initial states. Blue tokens represent the planned actions to be executed. Gray tokens are the effects of accomplishing the planned actions. The execution frontier is denoted by τ.

The current coordination process is an essential step which requires a future deeper assessment to provide ARIES a more robust execution against system failures. Currently, ARIES is just able to manage the communication lost and unachieved goal failures states, which is not enough to provide a reliable autonomous controller. A possible way to achieve a robust failure management system could be to deploy a new T-REX reactor on every ARIES agent (leader and follower(s)) that executes its own planner with the world knowledge to deal with a wide range of failures. The current coordinated execution will keep informed this reactor of every action execution, so it will be able to execute the suitable contingency plan.

The computational results of executing ARIES over the nominal scenario are presented in Table 5.1, showing the communications between agents for coordination (expressed in number of tokens exchanged), the charging stops performed by the UAV to reach the targets, the deliberation time required by UP2TA to obtain a plan, the simulation time taken by V-REP to perform the simulation and the distances travelled by the UGV and the UAV.

We can observe that ARIES performs 64 communications between the leader and the follower to accomplish the mission. As we can appreciate in the partial plan shown in Figure 5.14, 8 actions execution (blue boxes) in the follower agent require 8 effect tokens, so the complete nominal scenario with 32 actions requires a total of 64 communications. The UAV's high capacity battery allows the heterogeneous simple UGV-UAV system to reach the 6 targets by computing only 4 charging stops (green boxes in Figure 5.12a).

Table 5.1: Experimental results of executing ARIES in the nominal scenario and another three scenarios with different set-ups. #Com = number of communication between agents, #CS = number of charging stops, D_t = deliberation time required by UP2TA to obtain the plan, S_t = simulation time taken by V-REP to complete the simulation, F_{ugv} = distance travelled by the UGV end F_{uav} = distance travelled by the UAV.

	#Com	#CS	D_t (s)	S_t (s)	F_{ugv} (m)	F_{uav} (m)
Nominal	64	4	25.2	459.9	138.6	125.5
Nominal L-C	70	6	27.8	469.6	146.8	71.4
Extended	128	8	40.3	668.1	152.1	168.2
Extended L-C	146	11	44.2	761.5	158.2	52.4

Additionally, we execute ARIES over scenarios derived from the nominal scenario. The tested scenarios are described as follows:

- **Nominal with low-capacity** (Nominal L-C in Table 5.1). Starting at the home location, the objective is to take 6 pictures. The UAV has a lower battery capacity than the nominal, so it requires the UGV to travel more distance in order to deploy the UAV closer to the targets. Therefore, it is a slightly complex scenario than the nominal, and so, the planner takes more time to compute a plan.

- **Extended nominal** (Extended in Table 5.1). Starting at the home location, the objective is to take 12 pictures (six pictures more than the nominal). This scenario is a more complex than the previous one from the planning perspective as it is required to plan more targets.

- **Extended nominal with low-capacity** (Extended L-C in Table 5.1) It is similar to the previous one, but the UAV has a lower battery capacity. Thus, it represents a combination of the previous scenarios where the UGV needs to travel a longer distance to allow the UAV to reach the targets, while the planner has to deal also with a high number (12) of targets.

The results of executing the above scenarios are also presented in Table 5.1. First, we can observe that the communications and the charging stops are increased either by the task number or the battery capacity. Second, the deliberation and simulation times experiment an increment due to the scenario complexity. Finally, F_{ugv} is higher in the low-capacity scenarios because it is required to deploy the UAV closer to the targets to allow it to reach the objectives without running out of energy. This explains why F_{uav} is quite lower on the low-capacity scenarios.

5.5 SUMMARY

Over the last two decades, autonomous controllers have proven their efficacy to tackle problems in which robotic systems need a high level of autonomy. The algorithms presented in chapter 3 and chapter 4 just represent the decision-making process to overcome these problems, so there are required complex entities, such as these autonomous controllers, to execute and monitor those decisions. In this chapter, we presented a research effort for bringing up an autonomous controller with the ability to operate the cooperation of a multi-robot system. We named this controller as ARIES, and it has been designed on the pillars of T-REX, an autonomous controller originally designed by McGann et al. [263] to control one and/or more robots from a centralized single agent. In section 5.1, we reviewed and summarized the T-REX controller, which we took advantage of to build ARIES. For instance, we mentioned that T-REX follows the timeline-base paradigm to model the execution of a robotic agent, but in section 5.2 we explained why maybe it is better to use an action-based temporal paradigm instead. This is why ARIES uses this paradigm. In section 5.3, we explained the ARIES's key structural elements, its execution cycle, and the main properties we want to endow to our controller: flexibility, scalability and fault tolerant. We could summarize ARIES as a hierarchical system where the leader deliberates cooperative plans and coordinates the follower's execution, meanwhile each follower executes the received actions and notifies the outcomes to the leader. Due to ARIES is on its initial phase and the costs of conducting such experiments with real robots are substantial, the experimental demonstration has been carried out on a robotic simulator. Therefore, in section 5.4, we analysed a particular study case and the first results on different simulated scenarios.

Part III

THE CONCLUSIONS

6 CONCLUSIONS & FUTURE WORK

This chapter summarizes the conclusions of this dissertation following the order of the presented work. Also, the chapter ends by discussing some future interesting research lines that may extend the research here presented.

6.1 CONCLUSIONS

Throughout the dissertation, we endeavoured to expose the reader to the growing research interest in cooperative heterogeneous UGV-UAV systems to tackle historical problems, such as the exploration and last-mile delivery. As we have showed in the state of the art, this novel approach has experienced an explosion in the literature during the past decade because of the synergistic capabilities of these systems to overcome such problems. Driven by this principle, we carried out the research work explained at length in this dissertation.

We started our research by formulating an exploration problem in $\mathbb{R}^2$ Euclidean spaces, also called as the ECU-CSURP, that could be addressed by a heterogeneous simple UGV-UAV system. This exploration problem stands out because of the definition of a large-scale area where the robotics systems have neither enough energy nor functionalities to carry out the whole exploration, and so, they need to cooperate with the others to achieve the goals. In order to overcome this problem, we followed the existing UGV-UAV cooperation paradigm to design and implement a novel path planning algorithm. This path planning algorithm was named as TERRA, and it consists of splitting the exploration problem into five sub-problems (divide-and-conquer). In this way, TERRA devises a sequential set of stages, in which each stage solves a particular sub-problem, and then, it computes a cooperative path planning solution to the ECU-CSURP. We demonstrated the strong performance of the algorithm exploiting a cooperation paradigm in a heterogeneous simple UGV-UAV system. Also, we revealed that the farthest distance the UAV can travel and the clustering level, are the key features that need to be considered to maximize its performance. Furthermore, we extended TERRA for exploration problems in cumbersome terrains, such as high-hill or uneven areas. For that, we updated the ECU-CSURP to $\mathbb{R}^3$ Euclidean spaces, and then, we updated the first and third stages of TERRA and we added a new stage to compute the three-dimensional UGV's path. Therefore, we analysed the effects of executing TERRA in $\mathbb{R}^3$ scenarios from a safety perspective. We evaluated the performance searching for secure locations (or legit vertices) from where to reach the target points, and also, the performance with the new three-dimensional path planning.

We continued our research work by leveraging the TERRA algorithm to overcome the last-mile delivery problem with a heterogeneous multiple UGV-UAV system. This problem is being under study by the logistics companies since the beginning of the century. However, due to the exponential growth of deliveries experienced by the e-commerce companies, the last-mile delivery problem is nowadays a cutting-edge research topic. In this way, we formulated a generalization of the last-mile delivery problem, called as the mUCVLMP, which objective is to minimize the total delivery time under specific constraints that are not properly defined in the literature. Then, we introduced the task planning algorithm named as COURIER. This algorithm devises the same sequential set of stages than TERRA, but implements a completely new fifth stage that is able to compute a task planning solution. As well as the task planning capabilities, the main features of COURIER are the possibility of adding more UAVs to the solution and performing parallel deliveries by the UAVs, thanks to a novel geometrical rendezvous method. The experimental evaluation was carried out over a broad suite of instances with a different number of available UAVs and different UAV's configurations. Also, these configurations were tested for different number of parcels. The results showed a good behaviour, demonstrating that there is not need for expensive UAVs with a high capacity battery to cover a large area, and that each instance configuration presents a particular sub-optimal number of UAVs from which the objective function is minimized. Also, we analysed its performance by comparing it with the mFSTSP [179] approach. This approach deploys a similar heterogeneous multiple UGV-UAVs system, but it devises a different cooperation paradigm. This comparison demonstrated that the cooperation paradigm under study is not always the best solution, and even the UGV-UAV cooperation may be ineffective for some instances. But, it also showed a great potential because of the minimization of both the UGV and UAV travelling times, which could make it effective for real applications.

At this point, we demonstrated that the same cooperation paradigm integrated in TERRA and COURIER, present an overall good performance for both problems, which makes the heterogeneous (simple or multiple) UGV-UAV cooperation a feasible paradigm in computational terms. Nevertheless, these algorithms are not capable by themselves to fully take control of a robotic system. Instead, autonomous controllers are built for that purpose. In this direction, our research concludes with our efforts for designing and implementing a novel autonomous controller, called as ARIES, to exploit any kind of cooperation in a heterogeneous (simple or multiple) robotic system, such as the cooperation paradigm implemented by TERRA and COURIER. The ARIES controller is a decentralized architecture built on the existing TREX system. We can briefly define ARIES as a flexible, scalable and fault tolerant controller which follows a hierarchical scheme where the leader deliberates cooperative plans and coordinates the followers execution. We want to point out that ARIES represents our initial research effort to build a complete robust and feasible autonomous controllers. Even so, we evaluated the architecture in simulated scenarios, but with initial tests on real robotic platforms.

6.2 future lines of work

The work described in this dissertation suggests several extensions and directions for future work. Here, we review some possibilities for the work presented on each chapter.

Improving efficiency of TERRA for on-line path planning applications

Currently, the five stages of TERRA make it computationally efficient for off-line path planning, where an instantaneous response time is not required. Nevertheless, TERRA is not prepared for on-line path planning applications which demands a short response time. Even so, results show evidence of several ways to reduce the computational time of TERRA. One straightforward way could be to implement a branch-and-bound algorithm to solve the UAV's path in the fifth stage, instead of the search algorithm currently implemented. The literature about the TSP is extended, and it is encouraging the idea of evaluating the current performance against other branch-and-bound algorithms such as the Lin–Kernighan heuristic [141, 288] for the generalized symmetric and asymmetric TSP. As far as we know, the Lin–Kernighan heuristic is one of the strongest and successful heuristic for the TSP, so it could help TERRA to drastically reduce its computational time, and so, to be feasible for on-line path planning.

Multi-objective optimization against TERRA

We have shown that the ECU-CSURP consists on minimizing the UGV's and UAV's travelling distance. But, we can clearly observe that the ECU-CSURP presents a multi-objective optimization problem. The current TERRA algorithm addresses this problem from a single objective perspective. That is, it strategically executes a sequence of stages to minimize first with UGV's travelling distance, and secondly, the UAV's travelling distance. Aiming to explore another path to minimize the objective function of ECU-CSURP, it may be interesting the modelling of a new algorithm combining clustering capabilities with multi-objective optimization. That is, a new sequence of stages, where the first and second stages are solved with other clustering algorithms commonly used today for machine learning, such as k-means, and the next stages as a multi-objective optimization problem. Evolutionary algorithms [289] are a trending research topic for their good behaviour in multi-objective optimization problems. In this way, it would be interesting to evaluate the Pareto optimal solutions against the solutions given by TERRA.

A self-adaptive procedure of cooperation policies in COURIER

The geometrical rendezvous method and the arithmetic solver of COURIER describe a particular way of solving the time entanglements of vertices, that is, a particular cooperation policy to solve every complex combination of vertices. For instance, we have decided that the UAVs waiting for landing have the priority to land over the UAVs waiting for taking off in a shared vertex. Nevertheless, the current cooperation policy may not be the most efficient solution for all kind of time entanglements. In this way, it may be efficient to let the UAVs take off before the others land. Hence, an encouraging research line is to define an heuristic that helps the memetic algorithm to understand the problem, and then, choosing the most appropriated cooperation policy in each case. A hypothesis is that an algorithm with different cooperation policies available may outperform the current COURIER version with just one static policy.

Analysing COURIER for other related problems

At present, COURIER aims to solve the mUCVLMP, which is a generalization of the last-mile delivery problem. But, its robust and reliable design makes it suitable to address several well-known problems in the literature demanding heterogeneous multiple UGV-UAV systems. For instance, it's very attractive to us the possibility of adapting COURIER for vital supply in catastrophic areas, where a quick response is required to supply both medicine and food in hard access areas. Also, we would like to evaluate the performance of COURIER in search & rescue scenarios, where minimizing the time covering an area as well as the capabilities provided by the UGV-UAV cooperation paradigm may result very useful. Another possible application could be industrial or neighbourhood patrolling, where a group of UAVs along a UGV are able to persistently patrol a large area in minimal times. These are just some examples of numerous scenarios where COURIER might deliver a good performance.

Swarm behaviour among agents in ARIES

We like to think that leadership should be carefully taken into account. Sometimes, the leader can fall into the false premise that what he does is best for the team at the time. Here, we agree to reject this premise, and to believe that leadership should bring the capability to accept that co-workers may be right at some point. In some way, this dissertation promotes the cooperation as the most efficient way to address different problems. In this sense, it is encouraging integrating swarm behaviour into ARIES, which makes the robotic system following a desired collective behaviour following multi-objective purposes. Besides, the comparison between PDDL planning and swarm systems is a current research topic.

Part IV

APPENDIX

A EXPERIMENTAL SETTING FOR THE TERRA EVALUATION

In order to perform the evaluation of TERRA presented in section 3.4, a specific experimental setting was required. In the following, we present a random map algorithm to generate problem instances and a tuning stage where we set up the parameters of the genetic algorithm to obtain a reasonable performance during the experimental evaluation.

A.1 GENERATING RANDOM MAPS USING A GAUSSIAN DISTRIBUTION

We developed the random map generator showed in Algorithm 12 to build ECU-CSURP instances. A ECU-CSURP instance is a target point distribution which satisfies the problem statements (i-vi) described in section 3.1. Each distribution controls the location of the target points by the number of groups of closely located targets, i. e., by clusters. The location of the target points belonging to the same cluster is generated using a 2D Gaussian Distribution[1] with a random cluster center, and the parameters $\mu = 0$ and $\sigma = a * R$, where $a \in (0.5, 2)$ and R is the farthest distance the UAV can travel. Thus, we introduce a standard deviation in a range of $[R/2, 2R]$. The random number generation for the random center and a is controlled by the Suffle seed (based on the current time has been selected) and the Mersenne twister generator [290]. Therefore, our map generator relies on the following well defined parameters:

- N: number of target points.
- R: farthest distance the UAV can travel in km.
- δ: number of clusters.

The random map generator creates a map ensuring that N target points are distributed in δ clusters of N/δ target points following a 2D Gaussian Distribution inside an exploration area of a fixed size. Figure A.1 shows a feasible ECU-CSURP instance whose parameters satisfy the problem constraints. The (i) ECU-CSURP constraint is controlled by R. N and R represent the (ii) and (v) constraints respectively. The (iv) and (vi) constraints are controlled by placing a constant home location (V_0 in Figure A.1) for the whole experimentation. Also, the target points are distributed around an area satisfying the ECU-CSURP

1 https://es.mathworks.com/help/stats/normrnd.html

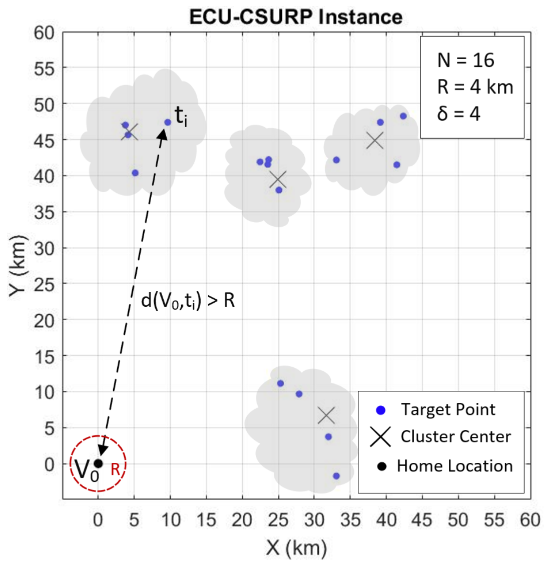


Figure A.1: A randomly generated ECU-CSURP instance. Note that the ECU-CSURP statements (i-vi) are satisfied.

distance constraint $\{t \in T : d(V_0, t) > R\} \neq \varnothing$ ((iii) constraint). Additionally, δ allows us to generate a broad random diversity of target points distributions.

Algorithm 12 Random Map Generator

```
Require: A, N, R, $\delta$
Ensure: T
 1: {$A$: Exploration area in $m^2$}
 2: {$T$: Set of target points}
 3: {$r_c$: Random cluster center}
 4: {$n_{cls}$: Number of clusters}
 5: $T \leftarrow \varnothing$
 6: $\sigma \leftarrow a * R$
 7: if $N > 0$ and $A > 0$ and $\delta > 0$ then
 8:     $n_{cls} \leftarrow N/\delta$
 9:     $rng(suffle, twister)$
10:     while $\delta > 0$ do
11:         $r_c \leftarrow [A * rand(1,1), A * rand(1,1)]$
12:         $a \leftarrow (1.5) * rand(1,1) + 0.5$
13:         $\sigma \leftarrow a * R$
14:         $t \leftarrow r_c + normrnd(0, sigma, [2, n_{cls}])$
15:         $T \leftarrow T \bigcup t$
16:         $\delta \leftarrow \delta - 1$
17:     end while
18: end if
19: return $T$
```

A.2 TUNING THE GENETIC ALGORITHM OF THE THIRD STAGE

The genetic algorithm described in subsection 3.2.4 requires a parameter tuning to allow us to characterise TERRA from a proper performance perspective. The objective is to select the best parameter setting minimizing the objective function 3.1 of the ECU-CSURP described in section 3.1. A parameter setting is formed by the parameters: population size (P_s), tournament size (T_s), mutation operator (M_o), mutation rate (M_r), crossover operator (C_o), crossover rate (C_r), elitism size (E_s). The mutation operator included into the parameter tuning are: Flip, Swap and Slide. Also, the crossover operators included are: Order Crossover (OX), Cycle Crossover (CX) and Order Base Crossover (CBX).

The tuning process starts computing a fixed budget of solution evaluations (*Seval*) to set a fair racing (which means to allocate the same resources) among different parameters settings. For example, an unfair racing would be to compare a parameter setting A with P_s = 500 with a parameter setting B with P_s = 100, both running the same number of generations (denoted as N_G). In this case, A is given five times more solutions evaluations than B, and therefore, A can perform a much wider exploration, which is unfair. Thus, we denoted $Seval = N_G * P_s$ to act as a stopping criterion of the genetic algorithm in the racing procedure. So, given a fixed *Seval*, the parameter setting A and B will run a proportional $N_G = Seval/P_s$ according to its P_s. Therefore, once *Seval* was fixed, the racing process could be launched to select the best parameter setting. The results of the experiment are publicly available on GitHub[2].

For the *Seval* computing, we generated one hundred random maps with $N = 16$ and one hundred random maps with $N = 64$. We set a standard parameter setting following the De Jong and Spears [84] guidelines: $P_s = 500$, $T_s = 4$, M_o = Flip, $M_r = 0.1$, C_o = CX, $C_r = 0.9$, $E_s = 1$. Then, we run the algorithm one time per random map, i. e., two hundred executions. Figure A.2 shows the results of this experiment. It plots the mean values (min, max, average) of the fitness function F_{ugv} on each generation, for random maps with $N = 16$ (black lines) and $N = 64$ (red lines). This convergence graph demonstrates that $N_G = 35$ is enough to compute the minimum fitness function. However, it may exist instances, with a similar parameter setting, whose convergence may be higher than 35 generations. In those cases, the algorithm would fail in finding the minimum fitness function. Consequently, we decided to set an offset until $N_G = 60$ generations, enough to minimize the objective function 3.1. Then, the fixed budget computed was $Seval = 60 * 500 = 30,000$ solution evaluations.

For the racing process, we used the *irace* package [291]. *irace* performs iterated racing procedures to automatically configure the genetic

2 https://github.com/FRopero/tuningGA

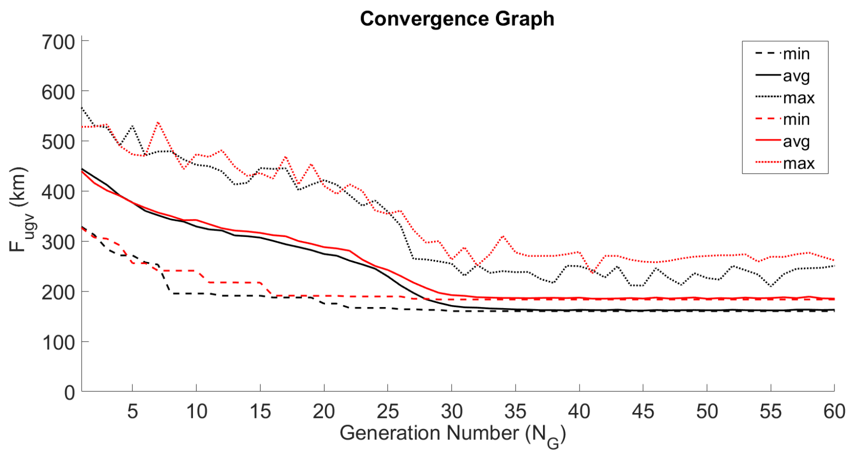


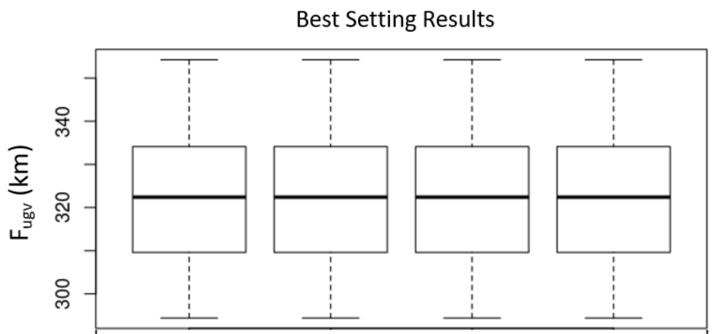
Figure A.2: Convergence graph plotting the mean values (min,avg,max) of F_{ugv} on each generation of the genetic algorithm. The black lines are random maps generated with $N = 16$, and the red lines with $N = 64$.

algorithm by finding the most appropriate setting, given a set of random map instances and a specific parameters space. The parameters space defines the range of allowed (and not allowed) values of each parameter. Table C.2 shows the chosen parameters space for this racing.

Table A.1: Parameters space of the genetic algorithm used for the *irace* tuning software. Percentages are referred to P_s.

Parameter	Type	Values
Ps	Integer	[100, 500]
Ts	Integer	{3, 6, 9, 12}
Mo	Enum	{Flip, Swap, Slide}
Mr	Float	[1.0%, 10.0%]
Co	Enum	{OX, CX, OBX}
Cr	Float	[90.0%, 99.0%]
Es	Float	[1.0%, 5.0%]

Then, *irace* performed a training experiment where it selected the best four appropriate settings. Once the training was finished, it run a testing experiment where the best four settings were computed. We generated one hundred random maps with $N = 16$ for training and one hundred more for testing. Figure A.3 shows at the bottom table, ordered from left to right, the best settings $ID = \{2, 82, 417, 646\}$ found by *irace*. At the top box-plot, Figure A.3 shows the results of executing each ID setting with each testing random map. We observe that the four settings have an equal performance and we can choose any of them to tune our genetic algorithm. In particular, all of them have a similar P_s, which means a similar N_G. However, the higher is P_s the higher are the selection pressure T_s and M_r. Also, we observe that a low selection pressure T_s is balanced with a high elitism E_s. Finally, we set up the genetic algorithm with the best setting $ID = 2$.



ID	2	82	417	646
Ps	430	452	450	456
Ts	9	12	9	12
Mo	Swap	Flip	Swap	Flip
Mr (%)	6.0	7.0	5.0	8.0
Co	OX	OX	OX	OX
Cr (%)	94.0	93.0	95.0	92.0
Es (%)	2.7	1.1	3.1	1.2

Figure A.3: Results for the training and testing experiment. At the bottom table, ordered from left to right, the best settings $ID = \{2, 82, 417, 646\}$ obtained in the training experiment. At the top box-plot, the *irace* results for computing the best settings in the testing experiment.

B ADDITIONAL EXPERIMENTS OF THE TERRA ALGORITHM

In this appendix we show additional statistical and computational experiments carried out to support the experiments presented in section 3.4.

B.1 STATISTICAL TESTS

One Way ANOVA tests were performed to demonstrate the statistical significance of Lemma 3.4.1 (see Table B.1) and Lemma 3.4.2 (see Table B.2) of Figure 3.8 in subsection 3.4.1. For the analysis, we applied the *anova1 function*[1] integrated in the MATLAB Statistics and Machine Learning Toolbox. Both tables show the probability value (p-value) obtained on each test. As a typical analysis, we used the standard significance level $\alpha = 0.05$ [292] to determine the statistical significance of the samples. For instance, a p-value $= 1.07 \times 10^{-17}$ in F_{ugv} and $\delta = 8$, in Table B.1, shows the One Way ANOVA test result for F_{ugv} with the three groups of samples $R = \{1, 3, 9\}$. More details are available on GitHub[2].

Table B.1: One Way ANOVA tests results (p-value) for F_{ugv}, F_{uav} and F_{total} with the three groups of samples $R = \{1, 3, 9\}$ and a fixed δ value on each column.

	$\delta = 8$	$\delta = 4$	$\delta = 2$
	$R = \{1, 3, 9\}$	$R = \{1, 3, 9\}$	$R = \{1, 3, 9\}$
F_{ugv}	1.07×10^{-17}	1.59×10^{-39}	1.88×10^{-34}
F_{uav}	0.00×10^{0}	0.00×10^{0}	0.00×10^{0}
F_{total}	7.38×10^{-307}	6.90×10^{-272}	8.76×10^{-205}

B.2 COMPUTATIONAL RESULTS

Extended computational experiments were performed for the TSPLib instances [252]: *bays29*, *eil51*, *eil76* and *berlin52* (see Table B.3). These instances have N = 29, 51, 76 and 52 respectively. We performed

1 https://es.mathworks.com/help/stats/anova1.html
2 https://github.com/FRopero/TERRA_Experiments

Table B.2: One Way ANOVA tests results (p-value) for F_{ugv}, F_{uav} and F_{total} with the three groups of samples $\delta = \{2,4,8\}$ and a fixed R value on each column.

	$R = 9$	$R = 3$	$R = 1$
	$\delta = \{2,4,8\}$	$\delta = \{2,4,8\}$	$\delta = \{2,4,8\}$
F_{ugv}	1.08×10^{-35}	9.19×10^{-92}	2.69×10^{-112}
F_{uav}	7.83×10^{-22}	2.67×10^{-13}	7.45×10^{-10}
F_{total}	2.87×10^{-43}	2.65×10^{-93}	6.91×10^{-110}

a computational study with $R = [4, 16, 64, 128]$ in km. The home location selected was the first target point displayed on each instance. The column headings are: **Name** (instance name), **Type** (weight type), **N** (number of target points) **R** (farthest distance the UAV can travel in km), F_{ugv} (UGV's distance travelled), F_{uav} (UAV's distance travelled), F_{total} (total distance travelled), **#C** (number of charging stops) and **Time** (computational time in seconds). The distance travelled in EUC instances is dimensionless, but the rest of them is in km, following the notation of the TSPLib.

Table B.3: Additional TERRA computational results for TSPLib instances.

Name	Type	N	R	F_{ugv}	F_{uav}	F_{total}	#C	Time
bays29	GEO	29	4	9535.1	224.0	9759.1	0	40.4
bays29	GEO	29	16	9679.8	656.1	10335.9	0	36.9
bays29	GEO	29	64	8927.0	3522.5	12439.5	0	37.3
bays29	GEO	29	128	8191.0	5708.4	13899.4	5	42.2
eil51	EUC	51	4	419.4	324.9	744.3	13	81.0
eil51	EUC	51	16	220.3	790.4	1010.7	20	335.7
eil51	EUC	51	64	0	729.3	729.3	5	9542.8
eil51	EUC	51	128	0	545.8	545.8	2	10279.0
eil76	EUC	76	4	510.1	542.7	1052.8	30	145.3
eil76	EUC	76	16	164.8	1143.5	1308.3	30	933.5
eil76	EUC	76	64	0	1279.6	1279.6	10	33333.0
eil76	EUC	76	128	0	746.9	746.9	2	40664.0
berlin52	EUC	52	4	10989.0	293.9	11282.9	0	84.8
berlin52	EUC	52	16	10393.0	1595.5	11988.5	3	83.4
berlin52	EUC	52	64	7757.9	4579.2	12337.1	14	110.6
berlin52	EUC	52	128	5758.8	8225.7	13984.5	20	178.0

C EXPERIMENTAL SETTING FOR THE COURIER EVALUATION

In this appendix we show additional aspects in the experimental set up for the evaluation of the COURIER algorithm presented in section 4.3. First, we show our extension of the standard TSPLib [252] to generate instances that can be used for the last-mile delivery problem. Second, we introduce the energy functions implemented in COURIER to compute the UAV's endurance in time and distance terms as are calculated by Murray and Raj [179], and so, to perform a fair comparison. And last, we describe the tuning stage carried out to extract the best parameter configuration for the memetic algorithm executed in the fifth stage of COURIER.

C.1 GENERATING TSPLIB INSTANCES FOR THE LAST-MILE DELIVERY PROBLEM

Nowadays, TSPLib instances are commonly used in the literature for any kind of generalizations of the TSP. In fact, it is a common practice that researchers extend this standard by introducing some particular parameters related to their particular problem. This is usually a good praxis because allows other researchers to evaluate their solutions against the others easily. Then, due to the current exponential growth in the last-mile delivery problem, we wanted to contribute to the efforts of creating such standard for this problem. Therefore, we created a standard to be able to execute instances by both algorithms in section 4.3. In the following, we describe the additional parameters introduced to create the extended format, and show an example.

- *CUSTOMER_SIZE* (integer): describes the number of parcels.
- *AREA_SIZE* (double) [m^2]: describes the size of the area.
- *DEPOT_LOCATION* (string): defines the depot location. The option we have used is *FIRST_NODE_COORD*, meaning that the first coordinate in the *NODE_COORD_SECTION* represents the depot location.
- *TRUCK_SPEED* (double) [m/s]: speed of the truck.
- *DRONES_NUMBER* (integer): number of available UAVs.
- *BATTERY_CAPACITY* (double) [j]: capacity of the battery in the UAVs.

- *PAYLOAD_CAPACITY* (double) [kg]: maximum weight of the parcel carried by the UAVs.
- *TAKEOFF_SPEED* (double) [m/s]: UAV's speed for the take off action.
- *FLYING_SPEED* (double) [m/s]: UAV's speed for the normal flying action.
- *LANDING_SPEED* (double) [m/s]: UAV's speed for the landing action.
- *FLIGHT_ALTITUDE* (double) [m]: altitude at which the UAVs will fly.
- *TAKEOFF_PACKAGE_CONSUME* (double) [$(W/(kg \cdot \frac{m}{s}))$]: consumption ratio of the UAVs during the take off stage carrying a parcel.
- *FLYING_PACKAGE_CONSUME* (double) [$(W/(kg \cdot \frac{m}{s}))$]: consumption ratio of the UAVs during the flying stage carrying a parcel.
- *LANDING_PACKAGE_CONSUME* (double) [$(W/(kg \cdot \frac{m}{s}))$]: consumption ratio of the UAVs during the landing stage carrying a parcel.
- *TAKEOFF_EMPTY_CONSUME* (double) [$(W/(\frac{m}{s}))$]: consumption ratio of the UAVs during the take off stage without a parcel.
- *FLYING_EMPTY_CONSUME* (double) [$(W/(\frac{m}{s}))$]: consumption ratio of the UAVs during the flying stage without a parcel.
- *LANDING_EMPTY_CONSUME* (double) [$(W/(\frac{m}{s}))$]: consumption ratio of the UAVs during the landing stage without a parcel.
- *SERVING_CUSTOMER_CONSUME* (double) [W]: power consumed by the UAVs delivering a parcel.
- *WAITING_TOLAND_CONSUME* (double) [W]: power consumed by the UAVs waiting for landing.
- *SERVICE_TIME* (double) [s]: time taken to deliver a parcel.
- *LAUNCH_TIME* (double) [s]: time taken to manually prepare a UAV for delivering (in case it is no fully autonomous)
- *RECOVERY_TIME* (double) [s]: time taken to manually recover a UAV (in case it is no fully autonomous).

The above parameters complete the extension created to generate instances of the last-mile delivery problem. Therefore, a possible TSPLib extended instance is displayed as shows Listing C.1.

Listing C.1: A TSPLib extended instance example.

```
CUSTOMER_SIZE: 10
AREA_SIZE: 600000000
DEPOT_LOCATION: FIRST_NODE_COORD
TRUCK_SPEED: 11
DRONES_NUMBER: 1
BATTERY_CAPACITY: 500000
PAYLOAD_CAPACITY: 5
TAKEOFF_SPEED: 15.6
FLYING_SPEED: 31.3
LANDING_SPEED: 7.8
FLIGHT_ALTITUDE: 50
TAKEOFF_PACKAGE_CONSUME: 11
FLYING_PACKAGE_CONSUME: 5.5
LANDING_PACKAGE_CONSUME: 11
TAKEOFF_EMPTY_CONSUME: 24
FLYING_EMPTY_CONSUME: 12
LANDING_EMPTY_CONSUME: 24
SERVING_CUSTOMER_CONSUME: 225
WAITING_TOLAND_CONSUME: 450
SERVICE_TIME: 60
LAUNCH_TIME: 60
RECOVERY_TIME: 30

NAME: N10A600U1HRHSG5-13
TYPE: TSP
COMMENT:
DIMENSION: 11
NODE_COORD_TYPE: TWOD_COORDS
DISPLAY_DATA_TYPE: NO DISPLAY
NODE_COORD_SECTION
0 0.5 0.5
1 14285.9193 15962.8645
2 3054.9104 16615.3197
3 6118.2725 11471.9719
4 2314.9767 14325.1381
5 20486.6988 15004.0793
6 16437.5189 20622.5049
7 7429.2655 19624.4376
8 14304.5491 23698.7242
9 13135.7456 7644.4759
10 8854.715 13421.2502
EOF
```

C.2 ENERGY FUNCTION IN COURIER

The energy functions designed for the experimental setting in section 4.3 are relevant to tackle a fair evaluation between the mFSTSP-h [179] and COURIER. These functions leverage the work of Dorling et al. [293] and Murray and Raj [179] devising a linear function depending

Table C.1: List of parameters used in the energy function to compute the UAV's endurance in distance and time terms for COURIER.

Name	Decription	Units
E_{avail}	Battery capacity	$[J]$
R	Farthest travelling distance	$[m]$
h	Flight altitude	$[m]$
to_s	Take off speed	$[m/s]$
c_s	Cruising speed	$[m/s]$
l_s	Landing speed	$[m/s]$
To	Take off time	$[s]$
Tf_a	Flying time to customer	$[s]$
Td	Service time	$[s]$
Tf_b	Flying time to UGV	$[s]$
Tl	Landing time	$[s]$
α_t	Consumption ratio in take off with parcel	$[(W/(kg \cdot \frac{m}{s}))]$
α_c	Consumption ratio in flying with parcel	$[(W/(kg \cdot \frac{m}{s}))]$
α_l	Consumption ratio in landing with parcel	$[(W/(kg \cdot \frac{m}{s}))]$
β_t	Consumption ratio in taking off without parcel	$[(W/(\frac{m}{s}))]$
β_c	Consumption ratio in flying without parcel	$[(W/(\frac{m}{s}))]$
β_l	Consumption ratio in landing without parcel	$[(W/(\frac{m}{s}))]$
Et	Power consumed in take off	$[J]$
Ef_a	Power consumed in flying to customer	$[J]$
Ed	Power consumed in the delivery	$[J]$
Ef_b	Power consumed flying back to the UGV	$[J]$
El	Power consumed landing	$[J]$

on the parcel weight, speed and operation time. We use this function to determine the UAV's endurance in time and distance terms as described in section 4.1. Table C.1 shows the list of parameters used in the functions.

First, we take the input parameters given at the experimental setting in subsection 4.3.1, and implement the following Equation C.1 to compute the UAV's endurance in distance terms. Here, given a particular battery capacity (E_{avail}), we compute the farthest travel distance for a UAV R as the UAV's endurance in distance terms, taking the payload capacity (w_p) as the maximum weight of a parcel as in Equation C.1.

$$
\begin{aligned}
To &= h/to_s \\
Tf_a &= R/c_s \\
Tf_b &= R/c_s \\
Tl &= h/l_s \\
Et &= To \cdot to_s \cdot (\alpha_t \cdot w_p + \beta_t) \\
Ef_a &= Tf_a \cdot c_s \cdot (\alpha_c \cdot w_p + \beta_c) \\
Ed &= Td \cdot \gamma \\
Ef_b &= Tf_b \cdot c_s \cdot \beta_c \\
El &= Tl \cdot l_s \cdot \beta_l \\
E_{avail} &= Et + Ef_a + Ed + Ef_b + El \qquad \text{(C.1)}
\end{aligned}
$$

Second, once we know R for a UAV (UAV's endurance in distance terms), we can use the previous equations to compute the maximum travel time T_{trip} for a UAV, which is the UAV's endurance in time terms. This is computed as follows:

$$
\begin{aligned}
To &= alt/to_s \\
Tf_a &= R/c_s \\
Tf_b &= R/c_s \\
Tl &= alt/l_s \\
T_{trip} &= To + Tf_a + Td + Tf_b + Tl \qquad \text{(C.2)}
\end{aligned}
$$

C.3 TUNING THE MEMETIC ALGORITHM

As well as the genetic algorithm of TERRA, the memetic algorithm described in section 4.2 (the fifth stage of the COURIER algorithm), demands a parameter tuning to perform a proper algorithm evaluation. Here, the objective is to select the best parameter setting that minimizes the objective Equation 4.1. In this case, a parameter setting is formed by the parameters: population size (P_s), tournament size (T_s), mutation rate (M_r), blind crossover rate (BC_r), heuristic crossover rate (HC_r). In the following, we provide an identical process to the genetic tuning described in Appendix A, so we avoid to exhaustively explain all the process for the shake of simplicity.

The first step is to compute a fixed budget of solution evaluations (*Seval*) to set a fair racing, as we did in the genetic tuning before. We already know that $Seval = N_G \cdot P_s$, where N_G is the number of generations. For the *Seval* computing we generated one hundred random maps with $N = 8$. We set a standard parameter setting following the same guidelines as in the genetic tuning: $P_s = 300$, $T_s = 4$, $M_r = 20\%$, $BC_r = 15\%$, and $HC_r = 10\%$. Figure C.1 shows the mean (min, avg and max) values of the fitness function T_{total}. This

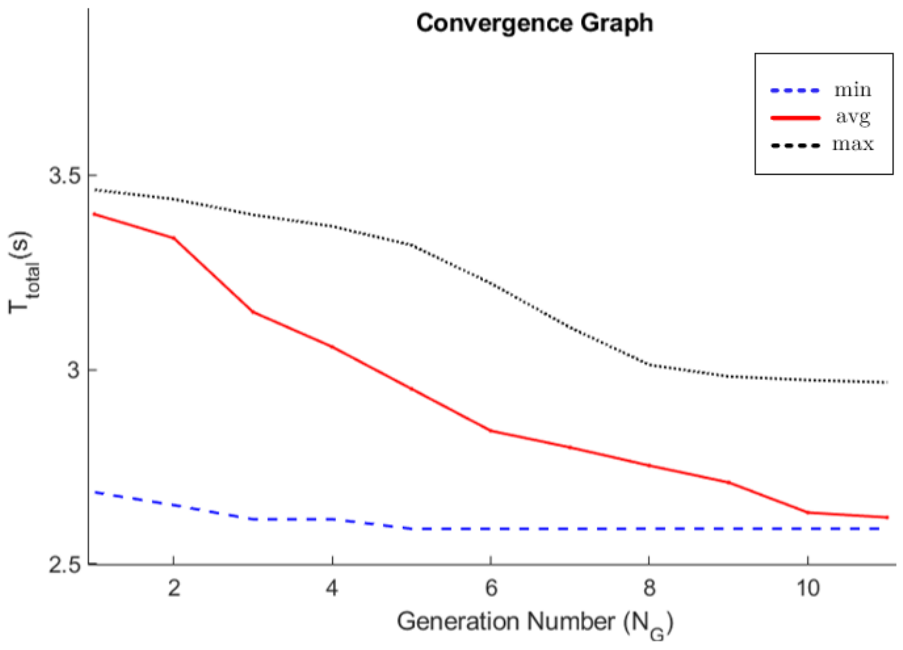


Figure C.1: Convergence graph plotting the mean values (min,avg,max) of T_{total} on each generation of the memetic algorithm.

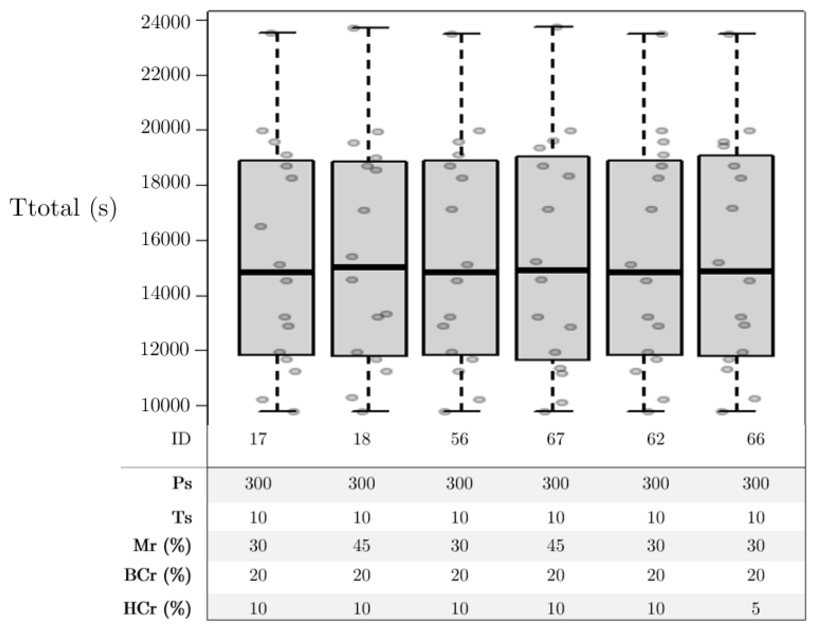
convergence graph demonstrates that $N_G = 10$ is enough to compute the minimum fitness function. Then, the computed fixed budget was $Seval = 10 \cdot 300 = 3000$ solution evaluations.

The second step is to perform the racing process to find the best configuration of parameters. As well, we used the *irace* package [291], which is in charge of finding the most appropriate setting, given a set of random map instances and a specific parameters space. Table C.2 shows the parameters space for this racing.

Table C.2: Parameters space of the memetic algorithm used for the *irace* tuning software. Percentages are referred to P_s.

Parameter	Type	Values
P_s	Integer	{100, 200, 300}
T_s	Integer	{5, 10, 15, 20}
M_r	Integer	{15%,30%,45%}
BC_r	Integer	{10%,20%,30%}
HC_r	Integer	{5%,10%,15%}

For the racing process, we run two procedures as well as we did for the genetic tuning. A first training procedure, where random instances are computed in order to get the best four configurations, and a second testing procedure, where the selected best six configurations were computed over random instances. Figure C.2 shows at the bottom table,



ID	17	18	56	67	62	66
Ps	300	300	300	300	300	300
Ts	10	10	10	10	10	10
Mr (%)	30	45	30	45	30	30
BCr (%)	20	20	20	20	20	20
HCr (%)	10	10	10	10	10	5

Figure C.2: Results for the training and testing experiment. At the bottom table, ordered from left to right, the best settings $ID = \{17, 18, 56, 67, 62, 66\}$ obtained in the training experiment. At the top box-plot, the *irace* results for computing the best settings in the testing experiment.

ordered from left to right, the best settings $ID = \{17, 18, 56, 67, 62, 66\}$ found by *irace*. At the top box-plot, Figure C.2 shows the results of executing each ID setting with each testing random map. We observe that the six settings have an equal performance and we can choose any of them to tune our genetic algorithm. In particular, all of them have $P_s = 300$, $T_s = 10, BC_r = 20\%$, and so, the same *Seval*. Also, we can observe that the main disparities come from the M_r and HC_r, but the mutation rate tends to increase whereas the heuristic crossover tends to decrease. We attribute this effect to the computational time differences between the two operators.

BIBLIOGRAPHY

[1] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. "Coordinated multi-robot exploration". In: *IEEE Transactions on robotics* 21.3 (2005), pp. 376–386.

[2] Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How. "Coordination and control of multiple UAVs". In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*. 2002.

[3] *Mars 2020 Mission Perseverance Rover with the Mars Helicopter Scout*. `https://mars.nasa.gov/mars2020/`. Accessed: 05-01-2020.

[4] Chase C Murray and Amanda G Chu. "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery". In: *Transportation Research Part C: Emerging Technologies* 54 (2015), pp. 86–109.

[5] Sergio Mourelo Ferrandez, Timothy Harbison, Troy Weber, Robert Sturges, and Robert Rich. "Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm". In: *Journal of Industrial Engineering and Management (JIEM)* 9.2 (2016), pp. 374–388.

[6] Phan Anh Tu, Nguyen Tuan Dat, and Pham Quang Dung. "Traveling salesman problem with multiple drones". In: *Proceedings of the Ninth International Symposium on Information and Communication Technology*. ACM. 2018, pp. 46–53.

[7] Dorit S Hochbaum and Wolfgang Maass. "Approximation schemes for covering and packing problems in image processing and VLSI". In: *Journal of the ACM (JACM)* 32.1 (1985), pp. 130–136.

[8] Refael Hassin and Nimrod Megiddo. "Approximation algorithms for hitting objects with straight lines". In: *Discrete Applied Mathematics* 30.1 (1991), pp. 29–42.

[9] Martin Grötschel, Alexander Martin, and Robert Weismantel. "The Steiner tree packing problem in VLSI design". In: *Mathematical Programming* 78.2 (1997), pp. 265–281.

[10] Michael R Bussieck, Thomas Winter, and Uwe T Zimmermann. "Discrete optimization in public rail transport". In: *Mathematical programming* 79.1-3 (1997), pp. 415–444.

[11] Michele Monaci. "Algorithms for packing and scheduling problems". In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1.1 (2003), pp. 85–87.

[12] Rochdi Sarraj, Eric Ballot, Shenle Pan, Driss Hakimi, and Benoit Montreuil. "Interconnected logistic networks and protocols: simulation-based efficiency assessment". In: *International Journal of Production Research* 52.11 (2014), pp. 3185–3208.

[13] Noam Nisan. "The communication complexity of approximate set packing and covering". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2002, pp. 868–875.

[14] Hidehisa Nakayama, Zubair Md Fadlullah, Nirwan Ansari, and Nei Kato. "A novel scheme for wsan sink mobility based on clustering and set packing techniques". In: *IEEE Transactions on Automatic Control* 56.10 (2011), pp. 2381–2389.

[15] Fatih Deniz, Hakki Bagci, Ibrahim Korpeoglu, and Adnan Yazıcı. "An adaptive, energy-aware and distributed fault-tolerant topology-control algorithm for heterogeneous wireless sensor networks". In: *Ad Hoc Networks* 44 (2016), pp. 104–117.

[16] Howie Choset. "Coverage for robotics–A survey of recent results". In: *Annals of mathematics and artificial intelligence* 31.1-4 (2001), pp. 113–126.

[17] Enric Galceran and Marc Carreras. "A survey on coverage path planning for robotics". In: *Robotics and Autonomous systems* 61.12 (2013), pp. 1258–1276.

[18] Onn Shehory and Sarit Kraus. "Task allocation via coalition formation among autonomous agents". In: *IJCAI*. 1995, pp. 655–661.

[19] Brian P Gerkey and Maja J Matarić. "A formal analysis and taxonomy of task allocation in multi-robot systems". In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.

[20] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. "Complexity of machine scheduling problems". In: *Annals of discrete mathematics*. Vol. 1. Elsevier, 1977, pp. 343–362.

[21] Vittorio Gorrini and Marco Dorigo. "An application of evolutionary algorithms to the scheduling of robotic operations". In: *European Conference on Artificial Evolution*. Springer. 1995, pp. 345–354.

[22] Hans Kellerer and Vitaly A Strusevich. "Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications". In: *Algorithmica* 57.4 (2010), pp. 769–795.

[23] Wojciech Bożejko, Andrzej Gnatowski, Ryszard Klempous, Michael Affenzeller, and Andreas Beham. "Cyclic scheduling of a robotic cell". In: *2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. IEEE. 2016, pp. 000379–000384.

[24] Dev Bahadur Poudel. "Coordinating hundreds of cooperative, autonomous robots in a warehouse". In: *Jan* 27 (2013), pp. 1–13.

[25] 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.

[26] Hana Godrich, Athina P Petropulu, and H Vincent Poor. "Sensor selection in distributed multiple-radar architectures for localization: A knapsack problem formulation". In: *IEEE Transactions on Signal Processing* 60.1 (2011), pp. 247–260.

[27] Gurkan Tuna, V Cagri Gungor, and Kayhan Gulez. "An autonomous wireless sensor network deployment system using mobile robots for human existence detection in case of disasters". In: *Ad Hoc Networks* 13 (2014), pp. 54–68.

[28] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. "A comprehensive taxonomy for multi-robot task allocation". In: *The International Journal of Robotics Research* 32.12 (2013), pp. 1495–1512.

[29] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. "Distributed algorithm design for multi-robot task assignment with deadlines for tasks". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3007–3013.

[30] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. "Multi-robot task allocation: A review of the state-of-the-art". In: *Cooperative Robots and Sensor Networks 2015*. Springer, 2015, pp. 31–51.

[31] Yang Yu and V. K. Prasanna. "Power-aware resource allocation for independent tasks in heterogeneous real-time systems". In: *Ninth International Conference on Parallel and Distributed Systems*. 2002, pp. 341–348.

[32] R. Liu, P. Sinha, and C. E. Koksal. "Joint Energy Management and Resource Allocation in Rechargeable Sensor Networks". In: *2010 Proceedings IEEE INFOCOM*. 2010, pp. 1–9.

[33] Gilbert Laporte. "The vehicle routing problem: An overview of exact and approximate algorithms". In: *European journal of operational research* 59.3 (1992), pp. 345–358.

[34] Tolga Bektas. "The multiple traveling salesman problem: an overview of formulations and solution procedures". In: *Omega* 34.3 (2006), pp. 209–219.

[35] Yves Crama, Vladimir Kats, Joris Van de Klundert, and Eugene Levner. "Cyclic scheduling in robotic flowshops". In: *Annals of operations Research* 96.1-4 (2000), pp. 97–124.

[36] Eugene Levner, Vladimir Kats, David Alcaide López de Pablo, and TC Edwin Cheng. "Complexity of cyclic scheduling problems: A state-of-the-art survey". In: *Computers & Industrial Engineering* 59.2 (2010), pp. 352–361.

[37] TC Edwin Cheng, Jatinder ND Gupta, and Guoqing Wang. "A review of flowshop scheduling research with setup times". In: *Production and operations management* 9.3 (2000), pp. 262–282.

[38] Sophie N Parragh, Karl F Doerner, and Richard F Hartl. "A survey on pickup and delivery problems". In: *Journal für Betriebswirtschaft* 58.1 (2008), pp. 21–51.

[39] Irina Dumitrescu, Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. "The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm". In: *Mathematical Programming* 121.2 (2010), p. 269.

[40] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[41] James J Kuffner and Steven M LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *IEEE International Conference on Robotics and Automation. Symposia Proceedings*. Vol. 2. IEEE. 2000, pp. 995–1001.

[42] Anna Yershova and Steven M LaValle. "Improving motion-planning algorithms by efficient nearest-neighbor searching". In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 151–157.

[43] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[44] Nikos Vlassis, Bas Terwijn, and Ben Krose. "Auxiliary particle filter robot localization from high-dimensional sensor observations". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation*. Vol. 1. IEEE. 2002, pp. 7–12.

[45] Jaewoo Chung, Matt Donahoe, Chris Schmandt, Ig-Jae Kim, Pedram Razavai, and Micaela Wiseman. "Indoor location sensing using geo-magnetism". In: *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM. 2011, pp. 141–154.

[46] GaoJun Fan and ShiYao Jin. "Coverage problem in wireless sensor network: A survey". In: *Journal of networks* 5.9 (2010), p. 1033.

[47] Krishnendu Chakrabarty, S Sitharama Iyengar, Hairong Qi, and Eungchun Cho. "Grid coverage for surveillance and target location in distributed sensor networks". In: *IEEE transactions on computers* 51.12 (2002), pp. 1448–1453.

[48] Santpal Singh Dhillon and Krishnendu Chakrabarty. "Sensor placement for effective coverage and surveillance in distributed sensor networks". In: *IEEE Wireless Communications and Networking. WCNC 2003*. Vol. 3. IEEE. 2003, pp. 1609–1614.

[49] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. "Autonomous UAV surveillance in complex urban environments". In: *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*. IEEE Computer Society. 2009, pp. 82–85.

[50] Denis Krivitski, Assaf Schuster, and Ran Wolff. "A local facility location algorithm for sensor networks". In: *International Conference on Distributed Computing in Sensor Systems*. Springer. 2005, pp. 368–375.

[51] Christian Frank and Kay Römer. "Distributed facility location algorithms for flexible configuration of wireless sensor networks". In: *International Conference on Distributed Computing in Sensor Systems*. Springer. 2007, pp. 124–141.

[52] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. "Coverage control for mobile sensing networks". In: *IEEE Transactions on robotics and Automation* 20.2 (2004), pp. 243–255.

[53] Jonathan Berry, William E Hart, Cynthia A Phillips, James G Uber, and Jean-Paul Watson. "Sensor placement in municipal water networks with temporal integer programming models". In: *Journal of water resources planning and management* 132.4 (2006), pp. 218–224.

[54] Xiaopeng Li and Yanfeng Ouyang. "Reliable sensor deployment for network traffic surveillance". In: *Transportation research part B: methodological* 45.1 (2011), pp. 218–231.

[55] Yael Edan, Tamar Flash, Uri M Peiper, Itzhak Shmulevich, and Yoav Sarig. "Near-minimum-time task planning for fruit-picking robots". In: *IEEE transactions on robotics and automation* 7.1 (1991), pp. 48–56.

[56] Sivakumar Rathinam, Raja Sengupta, and Swaroop Darbha. "A resource allocation algorithm for multivehicle systems with nonholonomic constraints". In: *IEEE Transactions on Automation Science and Engineering* 4.1 (2007), pp. 98–104.

[57] Ying-Fung Wu et al. "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles". In: *IEEE Transactions on Computers* 100.3 (1987), pp. 321–331.

[58] Ken Clarkson. "Approximation algorithms for shortest path motion planning". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM. 1987, pp. 56–65.

[59] Amna Khan, Iram Noreen, Hyejeong Ryu, Nakju Lett Doh, and Zulfiqar Habib. "Online complete coverage path planning using two-way proximity search". In: *Intelligent Service Robotics* 10.3 (2017), pp. 229–240.

[60] Way Kuo and Ming J Zuo. *Optimal reliability modeling: principles and applications*. John Wiley & Sons, 2003.

[61] Alan Cobham. "The Intrinsic Computational Difficulty of Functions". In: *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*. Ed. by Yehoshua Bar-Hillel. North-Holland Publishing, 1965, pp. 24–30.

[62] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158.

[63] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 29. wh freeman New York, 2002.

[64] Richard M Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[65] Juraj Hromkovič. *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*. Springer Science & Business Media, 2013.

[66] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[67] David S Johnson. "Approximation algorithms for combinatorial problems". In: *Journal of computer and system sciences* 9.3 (1974), pp. 256–278.

[68] Michel X Goemans and David P Williamson. "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming". In: *Journal of the ACM (JACM)* 42.6 (1995), pp. 1115–1145.

[69] David B Shmoys and KI Aardal. *Approximation algorithms for facility location problems*. Vol. 1997. Utrecht University: Information and Computing Sciences, 1997.

[70] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[71] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[72] Ronald A DeVore and Vladimir N Temlyakov. "Some remarks on greedy algorithms". In: *Advances in computational Mathematics* 5.1 (1996), pp. 173–187.

[73] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. "Globally-optimal greedy algorithms for tracking a variable number of objects". In: *2011 Conference on Computer Vision and Pattern Recognition*. IEEE. 2011, pp. 1201–1208.

[74] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[75] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT press, 2009.

[76] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA, 1995.

[77] Warren Buckler Powell. *Handbook of learning and approximate dynamic programming*. Vol. 2. John Wiley & Sons, 2004.

[78] Katta G Murty. *Linear programming*. Springer, 1983.

[79] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*. Vol. 6. Athena Scientific Belmont, MA, 1997.

[80] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[81] Michael O Rabin. "Probabilistic algorithm for testing primality". In: *Journal of number theory* 12.1 (1980), pp. 128–138.

[82] Pankaj K Agarwal and Micha Sharir. "Efficient randomized algorithms for some geometric optimization problems". In: *Discrete & Computational Geometry* 16.4 (1996), pp. 317–337.

[83] Michiel Smid. "Closest-point problems in computational geometry". In: *Handbook of computational geometry*. Elsevier, 2000, pp. 877–935.

[84] Kenneth A De Jong and William M Spears. "Using genetic algorithms to solve NP-complete problems." In: *ICGA*. 1989, pp. 124–132.

[85] Lawrence. Davis. "Handbook of Genetic Algorithms". In: 1990.

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

[87] Gerardo Beni and Jing Wang. "Swarm intelligence in cellular robotic systems". In: *Robots and biological systems: towards a new bionics?* Springer, 1993, pp. 703–712.

[88] James Kennedy. "Swarm intelligence". In: *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.

[89] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. "Distributed optimization by ant colonies". In: *Proceedings of the first European conference on artificial life*. Vol. 142. Cambridge, MA. 1992, pp. 134–142.

[90] Marco Dorigo and Mauro Birattari. *Ant colony optimization*. Springer, 2010.

[91] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984. ISBN: 0201055945.

[92] Yuval Davidor. *Genetic Algorithms and Robotics: A heuristic strategy for optimization*. Vol. 1. World Scientific, 1991.

[93] Osamu Takahashi and Robert J Schilling. "Motion planning in a plane using generalized Voronoi diagrams". In: *IEEE Transactions on robotics and automation* 5.2 (1989), pp. 143–150.

[94] Ellips Masehian and Davoud Sedighizadeh. "Classic and heuristic approaches in robot motion planning-a chronological review". In: *World Academy of Science, Engineering and Technology* 23.5 (2007), pp. 101–106.

[95] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. "Heuristic approaches in robot path planning: A survey". In: *Robotics and Autonomous Systems* 86 (2016), pp. 13–28.

[96] Xiaoming Zheng, Sonal Jain, Sven Koenig, and David Kempe. "Multi-robot forest coverage". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 3852–3857.

[97] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "Ffrob: An efficient heuristic for task and motion planning". In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 179–195.

[98] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. "Incremental Task and Motion Planning: A Constraint-Based Approach." In: *Robotics: Science and systems*. Vol. 12. Ann Arbor, MI, USA. 2016, p. 00052.

[99] Kelin Jose and Dilip Kumar Pratihar. "Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods". In: *Robotics and Autonomous Systems* 80 (2016), pp. 34–42.

[100] Donald Ervin Knuth. *The art of computer programming*. Vol. 3. Pearson Education, 1997.

[101] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. "An Algorithm for Finding Best Matches in Logarithmic Time". In: *ACM Trans. Math. Software* 3 (1977), pp. 209–226. DOI: 10.1145/355744.355745.

[102] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 891–923.

[103] K Fukunage and Patrenahalli M. Narendra. "A branch and bound algorithm for computing k-nearest neighbors". In: *IEEE transactions on computers* 7 (1975), pp. 750–753.

[104] Marius Muja and David Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." In: vol. 1. Jan. 2009, pp. 331–340.

[105] Roger Weber, Hans-Jörg Schek, and Stephen Blott. "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces". In: *VLDB*. Vol. 98. 1998, pp. 194–205.

[106] Alexandr Andoni and Piotr Indyk. "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions". In: *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE. 2006, pp. 459–468.

[107] Malcolm Slaney and Michael Casey. "Locality-sensitive hashing for finding nearest neighbors [lecture notes]". In: *IEEE Signal processing magazine* 25.2 (2008), pp. 128–131.

[108] Marius Muja and David G Lowe. "Scalable nearest neighbor algorithms for high dimensional data". In: *IEEE transactions on pattern analysis and machine intelligence* 36.11 (2014), pp. 2227–2240.

[109] Georges Voronoi. "Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs." In: *Journal für die reine und angewandte Mathematik* 134 (1908), pp. 198–287.

[110] Priyadarshi Bhattacharya and Marina L Gavrilova. "Voronoi diagram in optimal path planning". In: *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE. 2007, pp. 38–47.

[111] Dolores Blanco, Beatriz L Boada, and Luis Moreno. "Localization by voronoi diagrams correlation". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. Vol. 4. IEEE. 2001, pp. 4232–4237.

[112] L Chaimowicz, A Cowley, D Gomez-Ibanez, B Grocholsky, MA Hsieh, H Hsu, JF Keller, V Kumar, R Swaminathan, and CJ Taylor. "Deploying air-ground multi-robot teams in urban environments". In: *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 2005, pp. 223–234.

[113] Miguel García, Domenec Puig, Ling Wu, and Albert Solé. "Voronoi-Based Space Partitioning for Coordinated Multi-Robot Exploration". In: *Journal of Pysical Agents, ISSN 1888-0258* 1 (Jan. 2007), pp. 37–44. DOI: `10.14198/JoPha.2007.1.1.05`.

[114] Kai M Wurm, Cyrill Stachniss, and Wolfram Burgard. "Coordinated multi-robot exploration using a segmentation of the environment". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 1160–1165.

[115] Vasek Chvatal. "A greedy heuristic for the set-covering problem". In: *Mathematics of operations research* 4.3 (1979), pp. 233–235.

[116] László Lovász. "On the ratio of optimal integral and fractional covers". In: *Discrete mathematics* 13.4 (1975), pp. 383–390.

[117] Dorit S Hochbaum. "Approximation algorithms for the set covering and vertex cover problems". In: *SIAM Journal on computing* 11.3 (1982), pp. 555–556.

[118] Eran Halperin. "Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs". In: *SIAM Journal on Computing* 31.5 (2002), pp. 1608–1623.

[119] Kenneth L Clarkson and Kasturi Varadarajan. "Improved approximation algorithms for geometric set cover". In: *Discrete & Computational Geometry* 37.1 (2007), pp. 43–58.

[120] Reuven Bar-Yehuda and Shimon Even. "A linear-time approximation algorithm for the weighted vertex cover problem". In: *Journal of Algorithms* 2.2 (1981), pp. 198–203.

[121] Nabil H Mustafa and Saurabh Ray. "Improved results on geometric hitting set problems". In: *Discrete & Computational Geometry* 44.4 (2010), pp. 883–895.

[122] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. "A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem". In: *Science* 292.5516 (2001), pp. 472–475.

[123] William Rowan Hamilton. "Account of the icosian calculus". In: *Proceedings of the Royal Irish Academy*. Vol. 6. 1858, pp. 415–416.

[124] *A brief History of the Travelling Salesman Problem*. `https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/`. Accessed: 11-16-2019.

[125] Karl Menger. "Das botenproblem". In: *Ergebnisse eines mathematischen kolloquiums* 2 (1932), pp. 11–12.

[126] George Dantzig, Ray Fulkerson, and Selmer Johnson. "Solution of a large-scale traveling-salesman problem". In: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

[127] P Miliotis. “Integer programming approaches to the travelling salesman problem”. In: *Mathematical Programming* 10.1 (1976), pp. 367–378.

[128] W.L Eastman. *Linear programming with pattern constraints*. PhD thesis, Department of Economics, Harvard University, Cambridge, MA, 1958.

[129] John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. “An algorithm for the traveling salesman problem”. In: *Operations research* 11.6 (1963), pp. 972–989.

[130] Giorgio Carpaneto and Paolo Toth. “Some new branching and bounding criteria for the asymmetric travelling salesman problem”. In: *Management Science* 26.7 (1980), pp. 736–743.

[131] Ralph E Gomory. “An algorithm for integer solutions to linear programs”. In: *Recent advances in mathematical programming* 64.260-302 (1963), p. 14.

[132] Manfred Padberg and Giovanni Rinaldi. “A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems”. In: *SIAM review* 33.1 (1991), pp. 60–100.

[133] Martin Grötschel and Olaf Holland. “Solution of large-scale symmetric travelling salesman problems”. In: *Mathematical Programming* 51.1-3 (1991), pp. 141–202.

[134] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. *Concorde TSP solver*. 2006.

[135] Gilbert Laporte. “The traveling salesman problem: An overview of exact and approximate algorithms”. In: *European Journal of Operational Research* 59.2 (1992), pp. 231–247.

[136] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[137] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[138] Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis II. “An analysis of several heuristics for the traveling salesman problem”. In: *SIAM journal on computing* 6.3 (1977), pp. 563–581.

[139] Michel Gendreau, Alain Hertz, and Gilbert Laporte. “New insertion and postoptimization procedures for the traveling salesman problem”. In: *Operations Research* 40.6 (1992), pp. 1086–1094.

[140] Shen Lin. “Computer solutions of the traveling salesman problem”. In: *Bell System Technical Journal* 44.10 (1965), pp. 2245–2269.

[141] Shen Lin and Brian W Kernighan. "An effective heuristic algorithm for the traveling-salesman problem". In: *Operations research* 21.2 (1973), pp. 498–516.

[142] Ernesto Bonomi and Jean-Luc Lutton. "The N-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm". In: *SIAM review* 26.4 (1984), pp. 551–568.

[143] Bruce L Golden and Christopher C Skiscim. "Using simulated annealing to solve routing and location problems". In: *Naval Research Logistics Quarterly* 33.2 (1986), pp. 261–279.

[144] Fred Glover and Manuel Laguna. "Tabu search". In: *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[145] Mitsuo Gen and Lin Lin. "Genetic Algorithms". In: *Wiley Encyclopedia of Computer Science and Engineering* (2007), pp. 1–15.

[146] Donald Davendra. *Traveling Salesman Problem: Theory and Applications*. BoD–Books on Demand, 2010.

[147] Gilbert Laporte and Yves Nobert. "A cutting planes algorithm for the m-salesmen problem". In: *Journal of the Operational Research society* 31.11 (1980), pp. 1017–1023.

[148] A Iqbal Ali and Jeff L Kennington. "The asymmetric M-travelling salesmen problem: A duality based branch-and-bound algorithm". In: *Discrete Applied Mathematics* 13.2-3 (1986), pp. 259–276.

[149] Bezalel Gavish and Kizhanathan Srikanth. "An optimal solution method for large-scale multiple traveling salesmen problems". In: *Operations Research* 34.5 (1986), pp. 698–717.

[150] Robert A Russell. "An effective heuristic for the m-tour traveling salesman problem with some side conditions". In: *Operations Research* 25.3 (1977), pp. 517–524.

[151] George B Dantzig and John H Ramser. "The truck dispatching problem". In: *Management science* 6.1 (1959), pp. 80–91.

[152] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The vehicle routing problem: latest advances and new challenges*. Vol. 43. Springer Science & Business Media, 2008.

[153] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. "The vehicle routing problem: State of the art classification and review". In: *Computers & Industrial Engineering* 99 (2016), pp. 300–313.

[154] Yulun Wang and STEVENE Butner. "A new architecture for robot control". In: *Proceedings. 1987 IEEE International Conference on Robotics and Automation*. Vol. 4. IEEE. 1987, pp. 664–670.

[155] SY Nof and D Hanna. "Operational characteristics of multi-robot systems with cooperation". In: *The International Journal of Production Research* 27.3 (1989), pp. 477–492.

[156] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. “Cooperative mobile robotics: Antecedents and directions”. In: *Autonomous robots* 4.1 (1997), pp. 7–27.

[157] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. “A survey and analysis of multi-robot coordination”. In: *International Journal of Advanced Robotic Systems* 10.12 (2013), p. 399.

[158] Frankling Henry Giddings. *Sociology*. New York: Columbia University Press, 1908.

[159] Shin’ichi Yuta and Suparerk Premvuti. “Coordinating autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots”. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*. Vol. 3. IEEE. 1992, pp. 1566–1574.

[160] Ronald C Arkin and J David Hobbs. “Dimensions of communication and social organization in multi-agent robotic systems”. In: *Proceedings of the second international conference on From animals to animats*. Vol. 2. 1993, pp. 486–493.

[161] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.

[162] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. “Sensor planning for a symbiotic UAV and UGV system for precision agriculture”. In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1498–1511.

[163] Satyanarayana G Manyam, Kaarthik Sundar, and David W Casbeer. “Cooperative Routing for an Air-Ground Vehicle Team–Exact Algorithm, Transformation Method, and Heuristics”. In: *IEEE Transactions on Automation Science and Engineering* (2019).

[164] Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. “TERRA: A path planning algorithm for cooperative UGV–UAV exploration”. In: *Engineering Applications of Artificial Intelligence* 78 (2019), pp. 260–272.

[165] Lukas Klodt, Saman Khodaverdian, and Volker Willert. “Motion control for UAV-UGV cooperation with visibility constraint”. In: *2015 IEEE Conference on Control Applications (CCA)*. IEEE. 2015, pp. 1379–1385.

[166] M Barbier, H Cao, S Lacroix, C Lesire, F Teichteil-Königsbuch, and C Tessier. “Decision issues for multiple heterogeneous vehicles in uncertain environments”. In: *National conference on control architectures of robots (CAR)*. 2009.

[167] Gabriele Ferri and Vladimir Djapic. “Adaptive mission planning for cooperative autonomous maritime vehicles”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 5586–5592.

[168] Yu Wu. "Coordinated path planning for an unmanned aerial-aquatic vehicle (UAAV) and an autonomous underwater vehicle (AUV) in an underwater target strike mission". In: *Ocean Engineering* 182 (2019), pp. 162–173.

[169] Florian Shkurti, Anqi Xu, Malika Meghjani, Juan Camilo Gamboa Higuera, Yogesh Girdhar, Philippe Giguere, Bir Bikram Dey, Jimmy Li, Arnold Kalmbach, Chris Prahacs, et al. "Multi-domain monitoring of marine environments using a heterogeneous robot team". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1747–1753.

[170] P Valdivia y Alvarado, T Taher, H Kurniawati, G Weymouth, RR Khan, J Leighton, G Papadopoulos, G Barbastathis, and N Patrikalakis. "A coastal distributed autonomous sensor network". In: *OCEANS'11 MTS/IEEE KONA*. IEEE. 2011, pp. 1–8.

[171] Tyler Gunn and John Anderson. "Dynamic heterogeneous team formation for robotic urban search and rescue". In: *Journal of Computer and System Sciences* 81.3 (2015), pp. 553–567.

[172] Prithviraj Dasgupta, José Baca, KR Guruprasad, Angélica Muñoz-Meléndez, and Janyl Jumadinova. "The comrade system for multirobot autonomous landmine detection in postconflict regions". In: *Journal of Robotics* 2015 (2015).

[173] Gautham P Das, Thomas M McGinnity, Sonya A Coleman, and Laxmidhar Behera. "A distributed task allocation algorithm for a multi-robot system in healthcare facilities". In: *Journal of Intelligent & Robotic Systems* 80.1 (2015), pp. 33–58.

[174] Jeremi Gancet, Gautier Hattenberger, Rachid Alami, and Simon Lacroix. "Task planning and control for a multi-UAV system: architecture and algorithms". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 1017–1022.

[175] Neil Mathew, Stephen L Smith, and Steven L Waslander. "Multirobot rendezvous planning for recharging in persistent tasks". In: *IEEE Transactions on Robotics* 31.1 (2015), pp. 128–142.

[176] Kaarthik Sundar and Sivakumar Rathinam. "Algorithms for heterogeneous, multiple depot, multiple unmanned vehicle path planning problems". In: *Journal of Intelligent & Robotic Systems* 88.2-4 (2017), pp. 513–526.

[177] PB Sujit, Joao Sousa, and F Lobo Pereira. "UAV and AUVs coordination for ocean exploration". In: *Oceans 2009-Europe*. IEEE. 2009, pp. 1–7.

[178] Sathyaram Venkatesan. "AUV for Search & Rescue at sea-an innovative approach". In: *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*. IEEE. 2016, pp. 1–9.

[179] Chase C. Murray and Ritwik Raj. "The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones". In: *Transportation Research Part C: Emerging Technologies* 110 (2020), pp. 368 –398. ISSN: 0968-090X. DOI: `https://doi.org/10.1016/j.trc.2019.11.003`.

[180] Bingxi Li, Barzin Moridian, Anurag Kamal, Sharvil Patankar, and Nina Mahmoudian. "Multi-robot mission planning with static energy replenishment". In: *Journal of Intelligent & Robotic Systems* 95.2 (2019), pp. 745–759.

[181] A Aghaeeyan, Farzaneh Abdollahi, and Heidar Ali Talebi. "UAV–UGVs cooperation: With a moving center based trajectory". In: *Robotics and Autonomous Systems* 63 (2015), pp. 1–9.

[182] Sara Minaeian, Jian Liu, and Young-Jun Son. "Vision-based target detection and localization via a team of cooperative UAV and UGVs". In: *IEEE Transactions on systems, man, and cybernetics: systems* 46.7 (2015), pp. 1005–1016.

[183] Ben Grocholsky, James Keller, Vijay Kumar, and George Pappas. "Cooperative air and ground surveillance". In: *IEEE Robotics & Automation Magazine* 13.3 (2006), pp. 16–25.

[184] Barbara Arbanas, Antun Ivanovic, Marko Car, Matko Orsag, Tamara Petrovic, and Stjepan Bogdan. "Decentralized planning and control for UAV–UGV cooperative teams". In: *Autonomous Robots* 42.8 (2018), pp. 1601–1618.

[185] Boris Sofman, J Bagnell, Anthony Stentz, and Nicolas Vandapel. "Terrain classification from aerial data to support ground vehicle navigation". In: *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMURI-TR-05-39* (2006).

[186] L Cantelli, M Lo Presti, M Mangiameli, CD Melita, and G Muscato. "Autonomous cooperation between UAV and UGV to improve navigation and environmental monitoring in rough environments". In: *Proceedings 10th International symposium HUDEM,(ISSN 1848-9206)*. 2013, pp. 109–112.

[187] Elias Mueggler, Matthias Faessler, Flavio Fontana, and Davide Scaramuzza. "Aerial-guided navigation of a ground robot among movable obstacles". In: *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE. 2014, pp. 1–8.

[188] E. H. C. Harik, F. Guérin, F. Guinand, J. Brethé, and H. Pelvillain. "UAV-UGV cooperation for objects transportation in an industrial area". In: *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE. 2015, pp. 547–552.

[189] Martin Saska, Tomas Krajnik, and Libor Pfeucil. “Cooperative μUAV-UGV autonomous indoor surveillance”. In: *International Multi-Conference on Systems, Signals & Devices*. IEEE. 2012, pp. 1–6.

[190] Christopher Reardon and Jonathan Fink. “Air-ground robot team surveillance of complex 3D environments”. In: *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2016, pp. 320–327.

[191] Christos Papachristos and Anthony Tzes. “The power-tethered uav-ugv team: A collaborative strategy for navigation in partially-mapped environments”. In: *22nd Mediterranean Conference on Control and Automation*. IEEE. 2014, pp. 1153–1158.

[192] Parikshit Maini and PB Sujit. “On cooperation between a fuel constrained UAV and a refueling UGV for large scale mapping applications”. In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2015, pp. 1370–1377.

[193] Shannon Hood, Kelly Benson, Patrick Hamod, Daniel Madison, Jason M O’Kane, and Ioannis Rekleitis. “Bird’s eye view: Cooperative exploration by UGV and UAV”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 247–255.

[194] Kevin Yu, Ashish Kumar Budhiraja, Spencer Buebel, and Pratap Tokekar. “Algorithms and experiments on routing of unmanned aerial vehicles with mobile recharging stations”. In: *Journal of Field Robotics* 36.3 (2019), pp. 602–616.

[195] Luciano Cantelli, Michele Mangiameli, C Donato Melita, and Giovanni Muscato. “UAV/UGV cooperation for surveying operations in humanitarian demining”. In: *2013 IEEE international symposium on safety, security, and rescue robotics (SSRR)*. IEEE. 2013, pp. 1–6.

[196] Khaled A Ghamry, Mohamed A Kamel, and Youmin Zhang. “Cooperative forest monitoring and fire detection using a team of UAVs-UGVs”. In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2016, pp. 1206–1211.

[197] Neil Mathew, Stephen L Smith, and Steven L Waslander. “Planning paths for package delivery in heterogeneous multirobot teams”. In: *IEEE Transactions on Automation Science and Engineering* 12.4 (2015), pp. 1298–1308.

[198] Charles Pippin, Gary Gray, Michael Matthews, Dave Price, Ai-Ping Hu, Warren Lee, Michael Novitzky, and Paul Varnell. *The design of an air-ground research platform for cooperative surveillance*. Tech. rep. Georgia Tech Research Institute, 2010.

[199] Chad Hager, Dimitri Zarzhitsky, Hyukseong Kwon, and Daniel Pack. "Cooperative target localization using heterogeneous unmanned ground and aerial vehicles". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 2952–2957.

[200] Serkan Çaşka and Ahmet Gayretli. "An Unmanned Ground Vehicle-aided Task Exchange System Of Small Air Vehicles For Remote Surveillance Missions". In: *International Journal of Engineering Research and General Science* 4.2 (2016), pp. 873–883.

[201] Jin Hyo Kim, Ji-Wook Kwon, and Jiwon Seo. "Multi-UAV-based stereo vision system without GPS for ground obstacle mapping to assist path planning of UGV". In: *Electronics Letters* 50.20 (2014), pp. 1431–1432.

[202] Ignacio Mas, Patricio Moreno, Juan Giribet, and Diego Valentino Barzi. "Formation control for multi-domain autonomous vehicles based on dual quaternions". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 723–730.

[203] Jiao-Lin Shi, Bin Hu, Long Chen, Ding-Xue Zhang, Ding-Xin He, Jian Huang, and Zhi-Hong Guan. "Formation Tracking of Heterogeneous UGV-UAV Systems with Switching Directed Topologies". In: *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*. IEEE. 2019, pp. 970–975.

[204] Yang Chen, Shiwen Ren, Zhihuan Chen, Mengqing Chen, and Huaiyu Wu. "Path Planning for Vehicle-borne System Consisting of Multi Air–ground Robots". In: *Robotica* (2019), pp. 1–19.

[205] *Amazon Prime Air*. `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`. Accessed: 11-26-2019.

[206] *DHL Parcel Copter*. `https://www.dpdhl.com/en/media-relations/press-releases/2019/dhl-launches-its-first-regular-fully-automated-and-intelligent-urban-drone-delivery-service.html`. Accessed: 11-26-2019.

[207] *UPS Drone Delivery Service*. `https://www.ups.com/us/es/services/knowledge-center/article.page?kid=cd18bdc2`. Accessed: 11-26-2019.

[208] *FedEx Wing Express*. `https://qz.com/1712200/google-wing-launching-us-drone-deliveries-with-fedex-walgreens/`. Accessed: 11-26-2019.

[209] *Amazon's Jeff Bezos looks to the future*. `https://www.cbsnews.com/news/amazons-jeff-bezos-looks-to-the-future/`. Accessed: 11-26-2019.

[210] Niels Agatz, Paul Bouman, and Marie Schmidt. "Optimization approaches for the traveling salesman problem with drone". In: *Transportation Science* 52.4 (2018), pp. 965–981.

[211] Kai Peng, Jingxuan Du, Fang Lu, Qianguo Sun, Yan Dong, Pan Zhou, and Menglan Hu. "A hybrid genetic algorithm on routing and scheduling for vehicle-assisted multi-drone parcel delivery". In: *IEEE Access* 7 (2019), pp. 49191–49200.

[212] Lynne E Parker. "Current state of the art in distributed autonomous mobile robotics". In: *Distributed Autonomous Robotic Systems 4*. Springer, 2000, pp. 3–12.

[213] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. "Multirobot systems: a classification focused on coordination". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.5 (2004), pp. 2015–2028.

[214] Toshio Fukuda and Seiya Nakagawa. "A dynamically reconfigurable robotic system (concept of a system and optimal configurations)". In: *IECON'87: Industrial Applications of Robotics & Machine Vision*. Vol. 856. International Society for Optics and Photonics. 1987, pp. 588–595.

[215] Gerardo Beni. "The concept of cellular robotic system". In: *Proceedings IEEE International Symposium on Intelligent Control 1988*. IEEE. 1988, pp. 57–62.

[216] Hajime Asama, Akihiro Matsumoto, and Yoshiki Ishida. "Design of an autonomous and distributed robot system: ACTRESS". In: *Proceedings. IEEE/RSJ International Workshop on Intelligent Robots and Systems'.(IROS'89)'The Autonomous Mobile Robots and Its Applications*. IEEE. 1989, pp. 283–290.

[217] Alan H Bond and Les Gasser. *Readings in distributed artificial intelligence*. Morgan Kaufmann, 2014.

[218] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

[219] ESA-ESTEC Requirements & Standards Division. Space engineering: Space segment operability. *Technical Report ECSS-E-ST-70-11C, European Coordination for Space Standardization*. Tech. rep. Noordwijk, The Netherlands, 2008.

[220] Rodney Brooks. "A robust layered control system for a mobile robot". In: *IEEE journal on robotics and automation* 2.1 (1986), pp. 14–23.

[221] R James Firby. "An investigation into reactive planning in complex domains." In: *AAAI*. Vol. 87. 1987, pp. 202–206.

[222] Jacob T Schwartz and Micha Sharir. “On the “piano movers” problem. II. General techniques for computing topological properties of real algebraic manifolds”. In: *Advances in applied Mathematics* 4.3 (1983), pp. 298–351.

[223] Joseph S. B. Mitchell. “Planning Shortest Paths (Computational Geometry, Motion Planning, Visibility Graphs, Dijkstra’s Algorithm, Voronoi Diagrams)”. AAI8700791. PhD thesis. Stanford, CA, USA, 1986.

[224] Keiji Nagatani, Howie Choset, and Sebastian Thrun. “Towards exact localization without explicit localization with the generalized voronoi graph”. In: *1998 IEEE International Conference on Robotics and Automation*. Vol. 1. IEEE. 1998, pp. 342–348.

[225] Xiaoyu Yang, Mehrdad Moallem, and Rajnikant V Patel. “A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 35.6 (2005), pp. 1214–1224.

[226] Rajibul Huq, George KI Mann, and Raymond G Gosine. “Mobile robot navigation using motor schema and fuzzy context dependent behavior modulation”. In: *Applied soft computing* 8.1 (2008), pp. 422–436.

[227] Erann Gat, R Peter Bonnasso, Robin Murphy, et al. “On three-layer architectures”. In: *Artificial intelligence and mobile robots* 195 (1998), p. 210.

[228] Erann Gat. “Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots”. In: *AAAI*. Vol. 1992. 1992, p. 809.

[229] Rachid Alami, Raja Chatila, Sara Fleury, Malik Ghallab, and Félix Ingrand. “An architecture for autonomy”. In: *The International Journal of Robotics Research* 17.4 (1998), pp. 315–337.

[230] Richard Volpe, Issa Nesnas, Tara Estlin, Darren Mutz, Richard Petras, and Hari Das. “The CLARAty architecture for robotic autonomy”. In: *2001 IEEE Aerospace Conference Proceedings*. Vol. 1. IEEE. 2001, pp. 1–121.

[231] Nicola Muscettola, Gregory Dorais, Chuck Fry, Rich Levinson, and Christian Plaunt. “IDEA: Planning at the Core of Autonomous Reactive Agents”. In: (July 2002).

[232] Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. “T-rex: A model-based architecture for auv control”. In: *3rd Workshop on Planning and Plan Execution for Real-World Systems*. 2007.

[233] Luca Iocchi, Daniele Nardi, and Massimiliano Salerno. “Reactivity and deliberation: a survey on multi-robot systems”. In: *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*. Springer. 2000, pp. 9–32.

[234] Fabrice R Noreils. “Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment”. In: *The International Journal of Robotics Research* 12.1 (1993), pp. 79–98.

[235] Claude F Touzet. “Robot awareness in cooperative mobile robot learning”. In: *Autonomous Robots* 8.1 (2000), pp. 87–97.

[236] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. Addison-Wesley Reading, 1999.

[237] Philippe Caloud, Wonyun Choi, J-C Latombe, Claude Le Pape, and Mark Yim. “Indoor automation with many mobile robots”. In: *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. IEEE. 1990, pp. 67–72.

[238] Fang Tang and Lynne E Parker. “Asymtre: Automated synthesis of multi-robot task solutions through software reconfiguration”. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, pp. 1501–1508.

[239] Dejan Milutinovi and Pedro Lima. “Modeling and optimal centralized control of a large-size robotic population”. In: *IEEE Transactions on Robotics* 22.6 (2006), pp. 1280–1285.

[240] Ignacio Mas and Christopher Kitts. “Centralized and decentralized multi-robot control methods using the cluster space control framework”. In: *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. IEEE. 2010, pp. 115–122.

[241] Lynne E Parker. “ALLIANCE: An architecture for fault tolerant multirobot cooperation”. In: *IEEE transactions on robotics and automation* 14.2 (1998), pp. 220–240.

[242] Charles Lesire, Guillaume Infantes, Thibault Gateau, and Magali Barbier. “A distributed architecture for supervision of autonomous multi-robot missions”. In: *Autonomous Robots* 40.7 (2016), pp. 1343–1362.

[243] Kutluhan Erol, James Hendler, and Dana S Nau. “HTN planning: Complexity and expressivity”. In: *AAAI*. Vol. 94. 1994, pp. 1123–1128.

[244] Lynne E Parker, Balajee Kannan, Fang Tang, and Michael Bailey. “Tightly-coupled navigation assistance in heterogeneous multi-robot teams”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 1. IEEE. 2004, pp. 1016–1022.

[245] Assia Belbachir, Félix Ingrand, and Simon Lacroix. “A cooperative architecture for target localization using multiple AUVs”. In: *Intelligent service robotics* 5.2 (2012), pp. 119–132.

[246] *The Mars Helicopter Scout*. `https://www.youtube.com/watch?v=w3y7iJEe7uM`. Accessed: 11-12-2019.

[247] Lester E Dubins. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". In: *American Journal of mathematics* 79.3 (1957), pp. 497–516.

[248] David E Goldberg and Kalyanmoy Deb. "A comparative analysis of selection schemes used in genetic algorithms". In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93.

[249] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[250] Pablo Muñoz, María D. R-Moreno, and Bonifacio Castaño. "3Dana: A path planning algorithm for surface robotics". In: *Engineering Applications of Artificial Intelligence* 60 (2017), pp. 175–192.

[251] Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. "A Strategical Path Planner for UGV-UAV Cooperation on Mars Terrains". In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. (2018), pp. 106–118.

[252] Gerhard Reinelt. "TSPLIB—A traveling salesman problem library". In: *ORSA journal on computing* 3.4 (1991), pp. 376–384.

[253] David C Howell. *Statistical methods for psychology*. Cengage Learning, 2009.

[254] Pablo Moscato et al. "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". In: *Caltech concurrent computation program, C3P Report* 826 (1989).

[255] Terry Jones. "Evolutionary algorithms, fitness landscapes and search". PhD thesis. 1995.

[256] Pablo Moscato and Carlos Cotta. "A modern introduction to memetic algorithms". In: *Handbook of metaheuristics*. Springer, 2010, pp. 141–183.

[257] Carlos Cotta and José M Troya. "Embedding branch and bound within evolutionary algorithms". In: *Applied Intelligence* 18.2 (2003), pp. 137–153.

[258] Quang Huy Nguyen, Yew-Soon Ong, and Natalio Krasnogor. "A study on the design issues of memetic algorithm". In: *2007 IEEE Congress on Evolutionary Computation*. IEEE. 2007, pp. 2390–2397.

[259] Hans-Paul Schwefel. “Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of organic evolution”. In: *Annals of Operations Research* 1.2 (1984), pp. 165–167.

[260] Yuval Davidor and Oren Ben-Kiki. “The Interplay Among the Genetic Algorithm Operators: Information Theory Tools Used in a Holistic Way.” In: *PPSN*. Vol. 2. 1992, p. 75.

[261] Helen G Cobb and John J Grefenstette. *Genetic algorithms for tracking changing environments.* Tech. rep. Naval Research Lab Washington DC, 1993.

[262] *Amazon reveals new delivery drone design with range of 15 miles.* `https://www.geekwire.com/2015/amazon-releases-updated-delivery-drone-photos-video-showing-new-prototype/`. Accessed: 11-27-2019.

[263] Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. “A deliberative architecture for AUV control”. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 1049–1054.

[264] Kanna Rajan and Frédéric Py. “T-REX: partitioned inference for AUV mission control”. In: *Further advances in unmanned marine vehicles* (2012), pp. 171–199.

[265] N. Muscettola. “"HSTS: Integrating Planning and Scheduling"”. In: *Intelligent Scheduling*. Ed. by M. Zweben and M. Fox. San Francisco: Morgan Kaufrmann, 1994, pp. 169–212.

[266] Pablo Munoz and Marıa D. R-Moreno. “Deliberative systems for autonomous robotics: A brief comparison between action-oriented and timelines-based approaches”. In: *Proc. ICAPS Workshop Planning and Robotics*, pp. 45–54.

[267] Antonio Ceballos, Saddek Bensalem, Amedeo Cesta, L De Silva, Simone Fratini, Félix Ingrand, J Ocon, Andrea Orlandini, Frederic Py, Kanna Rajan, et al. “A goal-oriented autonomous controller for space exploration”. In: *ASTRA* 11 (2011).

[268] Marta Cialdea Mayer, Andrea Orlandini, and Alessandro Umbrico. “Planning and execution with flexible timelines: a formal account”. In: *Acta Informatica* 53.6-8 (2016), pp. 649–680.

[269] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini. “Timelines are expressive enough to capture action-based temporal planning”. In: *2016 23rd International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE. 2016, pp. 100–109.

[270] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini. “Complexity of timeline-based planning”. In: *Twenty-Seventh International Conference on Automated Planning and Scheduling*. 2017.

[271] Maria Fox and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research* 20 (2003), pp. 61–124.

[272] Sara Bernardini and David E Smith. "Translating pddl2.2 into a constraint-based variable/value language". In: *Proc. of the Workshop on Knowledge Engineering for Planning and Scheduling, 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. 2008.

[273] Stefan Edelkamp and Jörg Hoffmann. "PDDL2. 2: The language for the classical part of the 4th international planning competition". In: *4th International Planning Competition (IPC'04), at ICAPS'04* (2004).

[274] Jeremy Frank and Ari Jónsson. "Constraint-based attribute and interval planning". In: *Constraints* 8.4 (2003), pp. 339–364.

[275] Amedeo Cesta and Angelo Oddi. "DDL. 1: A formal description of a constraint representation language for physical domains". In: *New directions in AI planning* (1996), pp. 341–352.

[276] Pablo Muñoz, María D. R-Moreno, David F. Barrero, and Fernando Ropero. "MoBAr: A hierarchical action-oriented autonomous control architecture". In: *Journal of Intelligent & Robotic Systems* 94.3-4 (2019), pp. 745–760.

[277] Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. "ARIES: An Autonomous Controller For Multirobot Cooperation". In: *IEEE Aerospace and Electronic Systems Magazine* 34.3 (2019), pp. 40–55.

[278] Sara Fleury, Matthieu Herrb, and Raja Chatila. "GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture". In: *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS'97*. Vol. 2. IEEE. 1997, pp. 842–849.

[279] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009.

[280] Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. "A Versatile Executive Based on T-REX for Any Robotic Domain". In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. (2018), pp. 79–91.

[281] Pablo Muñoz, María D. R-Moreno, and David F. Barrero. "Unified framework for path-planning and task-planning for autonomous robots". In: *Robotics and Autonomous Systems* 82 (2016), pp. 1–14.

[282] Jörg Hoffmann and Bernhard Nebel. "The FF planning system: Fast plan generation through heuristic search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.

[283] Chih-Wei Hsu and Benjamin W Wah. "The SGPlan planning system in IPC-6". In: *Proceedings of the Sixth International Planning Competition*. 2008.

[284] Fernando Ropero, Daniel Vaquerizo, Pablo Muñoz, and María D. R-Moreno. "An Advanced Teleassistance System to Improve Life Quality in the Elderly". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. (2017), pp. 533–542.

[285] Fernando Ropero, Daniel Vaquerizo-Hdez, Pablo Muñoz, David F. Barrero, and Maria D. R-Moreno. "LARES: An AI-based teleassistance system for emergency home monitoring". In: *Cognitive Systems Research* 56 (2019), pp. 213–222.

[286] Eric Rohmer, Surya PN Singh, and Marc Freese. "V-REP: A versatile and scalable robot simulation framework". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1321–1326.

[287] Fernando Ropero, Pablo Munoz, María D. R-Moreno, and David F. Barrero. "A Virtual Reality Mission Planner for Mars Rovers". In: *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. IEEE. (2017), pp. 142–146.

[288] Keld Helsgaun. "An effective implementation of the Lin–Kernighan traveling salesman heuristic". In: *European Journal of Operational Research* 126.1 (2000), pp. 106–130.

[289] Gong Mao-Guo, Jiao Li-Cheng, Yang Dong-Dong, and Ma Wen-Ping. *Evolutionary multi-objective optimization algorithms*. 2009.

[290] Makoto Matsumoto and Takuji Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30.

[291] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[292] Regina Nuzzo. "Scientific method: statistical errors". In: *Nature News* 506.7487 (2014), p. 150.

[293] Kevin Dorling, Jordan Heinrichs, Geoffrey G Messier, and Sebastian Magierowski. "Vehicle routing problems for drone delivery". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.1 (2016), pp. 70–85.

DECLARATION

I declare that this thesis is an original report of my research, has been written by me and has not been submitted for any previous degree. The experimental work is almost entirely my own work; the collaborative contributions have been indicated clearly and acknowledged. Due references have been provided on all supporting literatures and resources.

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification

Alcalá de Henares, 2020

Fernando Ropero Pastor