UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

MASTER FINAL THESIS

Genetic Algorithms to Schedule Observations in Planetary Remote Sensing Missions

Document:

Report

Author:

Jorge Simón Aznar

Director/Co-director:

Director: Manel Soria

Co-Director: Paula Betriu

Degree:

Master in Space and Aeronautical Engineering

Examination session:

Extension 2023/2024

Abstract

Optimizing operations in space-related missions presents a unique challenge due to the limited resources available and the criticality of each action undertaken. Planetary missions, particularly in remote sensing, face significant constraints in terms of energy, time, and data acquisition windows, where every decision has a direct impact on the mission's success. Spacecraft must navigate complex operational environments, such as varying planetary geometries, unpredictable environmental conditions, and strict fuel and communication limitations. In missions like Jupiter Icy Moons Explorer (JUICE), where close flybys and short observation windows are the norm, the importance of precise optimization becomes even more pronounced. Failure to optimize the usage of instruments can lead to significant scientific losses.

This thesis focuses on the optimization of the observation schedule for the Jovis Amorum ac Natorum Undique Scrutator (JANUS) camera and Radar for Icy Moons Exploration (RIME) radar aboard JUICE, a European Space Agency (ESA) mission designed to explore Jupiter's moons. In this thesis, it is given a particular emphasis on Callisto. Given the limited flyby time and conflicting operational constraints, it is crucial to maximize the scientific return by optimizing the timing and duration of instrument usage during the mission's 12 flybys around Callisto from August 2032 to July 2033.

A Multi-Objective Genetic Algorithm (MOGA) is employed to address the scheduling problem, aiming to optimize two key objectives: maximizing image resolution for JANUS and maximizing surface coverage for RIME. The algorithm balances these objectives while satisfying geometric constraints and limited observation windows. By evolving a population of candidate schedules, the algorithm identifies a set of Pareto-optimal solutions, offering flexible trade-offs between competing scientific goals.

Results from two case studies demonstrate the effectiveness of the Genetic Algorithms (GAs) in generating diverse solutions, each balancing JANUS observations and RIME scanning. The first case examines a single flyby and Region Of Interest (ROI), while the second expands to multiple ROIs across four flybys. Visual and quantitative analysis of the Pareto fronts illustrates the trade-offs between the two objectives, showcasing the algorithm's capability to optimize conflicting mission requirements efficiently.

This research highlights the potential of MOGAs to enhance mission planning in space exploration, providing mission planners with optimized schedules that maximize scientific return while adhering to technical constraints. The ability to systematically explore trade-offs in such high-stakes, resource-limited environments is essential for the success of future planetary missions.

Optimizar las operaciones en misiones espaciales presenta un desafío único debido a los recursos limitados disponibles y la criticalidad de cada acción realizada. Las misiones planetarias, en particular las de teledetección, enfrentan restricciones significativas en términos de energía, tiempo y ventanas de adquisición de datos, donde cada decisión tiene un impacto directo en el éxito de la misión. Las naves espaciales deben navegar en entornos operacionales complejos, como geometrías planetarias variables, condiciones ambientales impredecibles y limitaciones estrictas de combustible y comunicación. En misiones como JUICE, donde los flybys cercanos y las ventanas de observación cortas son comunes, la importancia de la optimización precisa se vuelve aún más pronunciada. No optimizar el uso de los instrumentos puede conducir a pérdidas científicas significativas.

Esta tesis se centra en la optimización del calendario de observación para la cámara JANUS y el radar RIME a bordo de JUICE, una misión de la ESA diseñada para explorar las lunas de Júpiter. En este trabajo, se pone un énfasis particular en Calisto. Dado el tiempo limitado de los flybys y las restricciones operacionales conflictivas, es crucial maximizar el retorno científico optimizando el tiempo y la duración del uso de los instrumentos durante los 12 flybys de la misión alrededor de Calisto, desde agosto de 2032 hasta julio de 2033.

Se utiliza un MOGA para abordar el problema de la planificación, con el objetivo de optimizar dos objetivos clave: maximizar la resolución de imágenes para JANUS y maximizar la cobertura superficial para RIME. El algoritmo equilibra estos objetivos mientras satisface las restricciones geométricas y las ventanas de observación limitadas. Al hacer evolucionar una población de calendarios candidatos, el algoritmo identifica un conjunto de soluciones Pareto-óptimas, ofreciendo compensaciones flexibles entre los objetivos científicos en conflicto.

Los resultados de dos casos de estudio demuestran la efectividad de los GAs para generar soluciones diversas, equilibrando las observaciones de JANUS y el escaneo de RIME. El primer caso examina un solo flyby y una ROI, mientras que el segundo se expande a múltiples ROIs a lo largo de cuatro flybys. El análisis visual y cuantitativo de los frentes de Pareto ilustra las compensaciones entre los dos objetivos, mostrando la capacidad del algoritmo para optimizar eficazmente los requisitos conflictivos de la misión.

Esta investigación destaca el potencial de los MOGAs para mejorar la planificación de misiones en la exploración espacial, proporcionando a los planificadores de misiones calendarios optimizados que maximizan el retorno científico mientras se adhieren a las restricciones técnicas. La capacidad de explorar sistemáticamente estas compensaciones en entornos de alto riesgo y recursos limitados es esencial para el éxito de futuras misiones planetarias.

Acknowledgements

In this short page, I would like to thank Manel Soria, Paula Betriu, Pablo Torres and David De La Torre for their help and support in the development of this thesis.

In addition, I would also like to give a big thanks to my friends from the master's program; Iñaki Fernández, Antoni Escobosa, Marc Teixidor and Moritz Eisenbach for accompanying me during this year, even though I had to move to Madrid in the middle of the course, since without their help it would have been more complicated.

Also huge thanks to my family, my mum, my dad, my brother and Marina, who have been supporting me throughout the process, who have given me peace of mind when the economic situation was not the best and who have given me confidence in myself to overcome this stage. Finally, many thanks to my grandmother, one of the most important people in my life and who, although she has recently passed away, will always continue to guide and inspire me to overcome what lies ahead in my personal and professional life.

Contents

List of Figures

List of Tables

List of Abbreviations

GAs Genetic Algorithms

JUICE Jupiter Icy Moons Explorer

EAs Evolutionary Algorithms

MOGA Multi-Objective Genetic Algorithm

PS Particle Swarm

TS Tabu Search

RIME Radar for Icy Moons Exploration

DOF Degrees of Freedom

ESA European Space Agency

ESAC European Space Astronomy Centre

JPL Jet Propulsion Laboratory

PMOT Genetic Algorithm Toolkit

PE Planetary Exploration

JANUS Jovis Amorum ac Natorum Undique Scrutator

UV Ultraviolet

NIR Near Infrared

FOV Field of View

IFOV Instantaneous Field of View

ROI Region Of Interest

PSOA Python Science Opportunity Analysis

Chapter 1

Introduction

The exploration of our solar system through robotic missions has greatly enhanced our understanding of planetary bodies [1, 2, 3]. These missions are equipped with advanced remote sensing instruments, providing valuable data on celestial bodies' surfaces, compositions and structures. They have the potential to solve mysteries and offer insights that are not possible to obtain from observations on Earth. However, the success of these missions depends on the effective planning and scheduling of scientific observations. This task is complex due to the need to balance various constraints and objectives and the great number of possible combinations.

In planetary remote sensing missions, the science payload comprises various instruments designed for specific observational tasks. These instruments must operate within the limitations imposed by the spacecraft's resources, such as power, memory, and data downlink capabilities. The challenge is to optimize the use of these resources to maximize the scientific return [4]. Furthermore, there are also limitations in the observation availability, since it requires specific observation geometries. This involves selecting the most valuable observation opportunities from a vast array of possibilities. This process requires careful consideration of the mission trajectory, the environmental conditions affecting sensor performance, and the scientific objectives associated with each instrument.

Traditional planning and scheduling methods involve iterative cycles of input and refinement between instrument teams and mission planners. These methods, while effective, are time-consuming and heavily reliant on human expertise. The development of advanced computational techniques has created new opportunities for automating and optimizing this process [5, 6]. Specifically, the use of GAs offers a promising approach to address the multi-objective optimization challenges associated with scheduling planetary remote sensing observations [7].

The focus of this master's thesis is to explore the use of genetic algorithms in developing an automated observation planning and scheduling system for complex planetary missions. The methodology aims to generate optimal or near-optimal schedules that take into account all mission and instrument-specific constraints, thereby reducing the time and effort required for manual planning. The approach is validated through a case

study involving the JUICE mission.

The JUICE mission [8], is one of the most ambitious endeavours of the ESA, showcasing the challenges of planning missions for planetary exploration. With multiple flybys and an orbital phase around Jupiter's moon Ganymede, the mission's science operations need to be carefully planned to ensure high-quality data is captured within the limited observation windows available. This case study emphasizes the potential of genetic algorithms to manage the complex constraints and objectives involved in such missions, providing a robust solution for optimizing the scientific return.

This work is part of ongoing research aimed at creating comprehensive optimization tools for space mission planning, incorporating advances in algorithm design and computational techniques. The Genetic Algorithm Toolkit (PMOT) used in this study, developed by Professor Manel Soria, is integrated with NASA's Jet Propulsion Laboratory (JPL) SPICE toolkit [9] to manage mission data, demonstrating the practical application of these tools in a real-world context.

In conclusion, this thesis not only presents a new approach to scheduling observations in planetary remote sensing missions but also paves the way for future developments in space mission optimization. The integration of advanced algorithms and computational resources holds the promise of unlocking new frontiers in space exploration.

1.1 Objectives

This Master's Thesis has been developed under the supervision of Professor Manel Soria and Paula Betriu, and has three main objectives:

- To explore different methodologies in MOGA in order to improve the efficiency and performance of the algorithm.
- To implement this type of optimization algorithm in a real space-related problem.
- To demonstrate the feasibility of using these algorithms and their potential for future missions.

1.2 Scope

The Master's Thesis is divided into eight main chapters:

- In Chapter 2 the state of the art of the scheduling tools used throughout the evolution of space exploration missions are presented. In addition, the ESA's JUICE mission is introduced with an explanation of the main goals of the mission and the instruments that will be used for the development of this thesis.
- In Chapter 3, a brief introduction to GAs and MOGA is found. Then, different benchmarking tests are carried out to measure the performance of the implemented methods.

- In Chapter 4, the scheduling problem is shown, introducing the main concepts that characterized this problem and how the evolutionary operators have been coded.
- In Chapter 5, two different cases are developed to demonstrate the adaptability of the algorithm to the scheduling problem and the quality of the results.
- In Chapter 6, the Master's Thesis budget is presented.
- In Chapter 7, the environmental analysis and the social impact of the thesis is introduced.
- Finally, in Chapter 8, the conclusions of the developed work during the Master's Final Thesis are commented on.

Although this work has a broad context, there are certain topics that are out of the scope:

- A further testing of parameter settings and possible combinations to improve the performance of the algorithm. Only a few parameters are modified in the search for the most optimal performance.
- The different phases of the mission once the tour starts in Jupiter. Only the 12 flybys over Callisto are taken into account.
- Observations on bodies other than Callisto, directly related to the previous point.
- The design of the spacecraft trajectories, these are extracted directly from the SPICE kernels.

1.3 Requirements

The overall requirements of this Master's Thesis are the following ones:

- The code should be implemented in Python, chosen for its flexibility, wide range of applications, and open-source availability.
- Object-oriented programming should be used wherever applicable. Python's class structure ensures the code is more modular and easier to update or modify, allowing future functionalities to be added or overridden seamlessly.
- It must be adaptable, supporting different ROIs, celestial bodies, targets, or mission types.
- The code needs to be straightforward, both in readability and in how it is organized, to facilitate understanding and future development.

1.4 Justification

The optimization of observation scheduling for JUICE's flybys over Callisto using a MOGA presents a significant step forward in demonstrating the potential of these methods in space mission planning. The ESA's Consolidated Report on Mission Analysis (CReMA), specifically version 5.1 b2, which includes the most up-to-date and detailed trajectory data, provides an invaluable opportunity for this project. The availability of this comprehensive dataset, containing precise information about JUICE's flybys, spacecraft dynamics, and observation constraints, allows the development and application of advanced algorithms to solve real-world scheduling problems effectively.

Moreover, the success of this project will demonstrate the utility of MOGAs for both academic research and operational planning in space exploration. By providing optimized solutions that can adapt to changing mission profiles, such as those reflected in the iterative updates of CReMA, the work done here showcases the adaptability and robustness of MOGAs for space-related mission scheduling, potentially influencing future mission planning tools and approaches at ESA and other space agencies. Thus, this thesis not only explores the use of genetic algorithms for a complex scheduling problem but also paves the way for their broader application in real-world mission scenarios.

Chapter 2

State of the Art

The meticulous analysis of operations, especially the observations conducted by scientific instruments, is essential for the success of any satellite mission. This becomes even more critical in Planetary Exploration (PE) missions due to the challenging and complex environments in which these missions operate. Planning and scheduling operations in PE missions is a demanding and time-consuming process that still heavily relies on human expertise [10]. While there are several tools used in past and current missions, most are designed for detailed analysis of candidate operational schedules rather than for the automated and optimized creation of these schedules.

Several tools have been developed to assist in the planning and scheduling of satellite missions, each with its own set of capabilities and limitations, such as MAPPS [11], which is predominantly used to assist in medium- and short-term planning cycles. This tool excels in processing mission- and instrument-specific constraints, such as instrument pointing, illumination conditions, and potential environmental interferences, along with spacecraft trajectory data and preliminary operation schedules. The output includes a visualization of the schedule, highlighting any constraint violations, and provides a modifiable schedule that can be fine-tuned by operators. Additionally, this tool can convert scheduled science operations into command sequences necessary for executing observations. For example, MAPPS can simulate input schedules, offering a detailed profile of resource consumption over time due to the planned science operations and providing 2D projection maps of geometric features of the observations relative to the target body's surface.

SciBox [12], in development since 2001, shares similar visualization capabilities but also introduces an automatic schedule-building method. This method begins by identifying observation opportunities related to specific objectives and ranks them based on quality metrics, such as scene illumination and data resolution, derived from spacecraft trajectory and pointing data. Observations are then sequentially added to the schedule based on their rank, following a greedy strategy until a constraint is violated, thus generating an initial feasible observation schedule for further manual refinement. SciBox has been successfully applied in missions like MESSENGER, where it significantly improved data retrieval and command operations planning and, although it is automated in terms of generating feasible schedules, it is not a fully optimized schedule

generator.

Another tool, MAPGEN [13], developed by NASA Ames Research Center and the JPL, also provides timeline visualization and feasible schedule generation for top-level science operations in planetary missions. Leveraging the APGEN [14] engine, MAPGEN translates top-level operations into lower-level activities based on an activity dictionary, ensuring a comprehensive analysis of the impact on resource consumption and constraint compliance. Like SciBox, MAPGEN uses a simple greedy strategy to build schedules, sorting observation opportunities by priority scores specified by the user and selecting the highest-ranked opportunities that do not violate any constraints.

While SciBox and MAPGEN offer schedule generation capabilities, they primarily focus on constraint handling rather than maximizing the overall scientific return. The task of optimizing the observation schedule for each instrument to maximize the mission's value is largely left to human expertise.

In contrast, tools like ASPEN [15] or the CASPER [16] system go a step further by not only ensuring feasibility but also iteratively improving the schedule based on defined metrics or through iterative repair techniques. These approaches are more aligned with what is typically considered "optimization" in scheduling where the goal is to find the best possible schedule according to multiple objectives, rather than just a feasible one.

ASPEN, developed by the Jet Propulsion Laboratory, also focuses on constraint handling and automatically repairs any violations in a pre-built acquisition schedule using a method called iterative repair. This technique identifies and addresses individual constraint violations in the input schedule, modifying, moving, or eliminating operations as needed to produce a feasible schedule. ASPEN also includes a submodule for improving the quality of the input schedules by addressing activities with low preference values, although this approach is local and does not guarantee an optimized overall schedule quality.

CASPER, also developed at JPL, operates autonomously, making real-time adjustments to mission plans as the spacecraft receives new information or encounters unexpected events. CASPER uses iterative repair to generate and replan schedules on the fly, making it highly effective for real-time onboard mission planning. It enables spacecraft to autonomously handle unforeseen situations, adjusting its schedule to maximize science returns based on the latest data.

Eagle Eye [17] provides a comprehensive approach to mission planning, particularly for observations. Eagle Eye evaluates observational coverage, ensuring that observation windows are properly utilized. It is adapted to consider 2D framing instruments and the implicit area coverage planning problem. The latter involves finding the best possible coverage path of the instrument to maximize coverage over a ROI with the minimum possible make-span.

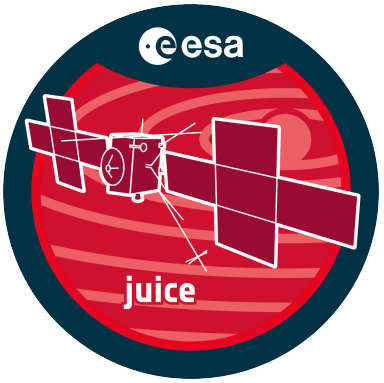
Finally, CLASP [18] adds another layer of sophistication. It uses Squeaky Wheel Optimization, an approach similar to CASPER's iterative repair method, but is designed to manage complex observation campaigns. It adjusts problematic tasks first, ensuring that key scientific objectives are met while handling geometric, temporal, and resource constraints. CLASP's integration with SPICE and its ability to visualize results using

Google Earth make it a powerful tool for mission planning, as demonstrated in missions like Europa Clipper and JUICE.

2.1 JUpiter ICy moons Explorer (JUICE)

JUICE is a groundbreaking mission led by the ESA as part of its Cosmic Vision 2015-2025 program. Selected in May 2012 as the program's first large-scale mission, JUICE was launched on April 14, 2023, and is expected to reach Jupiter by July 2031. For at least four years, the 3-axis stabilised spacecraft will conduct the most comprehensive exploration of the Jovian system[1] to date, with a particular focus on the gas giant and three of its largest moons: Ganymede, Europa, and Callisto [20].

Figure 2.1: ESA's JUICE patch [19].



The primary objective of JUICE is to investigate the conditions that might have supported the emergence of habitable environments on these icy satellites, particularly focusing on their subsurface oceans. Ganymede, the largest of Jupiter's moons and the only one with its own magnetic field will receive the most detailed study. This moon offers a natural laboratory for analyzing the nature, evolution, and potential habitability of icy worlds, as well as its interactions with Jupiter's magnetosphere. The mission aims to determine the characteristics of the subsurface oceans beneath the icy crusts of these moons, understand the sources and cycles of chemical and thermal energy, and explore the evolution of their surfaces and internal structures.

JUICE's investigation is critical for understanding the long-term stability of these environments and their potential to support life. By studying the gravitational interactions between the Galilean satellites and their tidal influences on the entire Jovian system, the mission will offer insights into the processes that have shaped these moons over geological timescales. Furthermore, JUICE will expand our understanding of Jupiter's atmosphere and magnetosphere, including their interactions with the satellites, thus deepening our knowledge of the dynamics and evolution of the entire Jovian system.

This mission stands as a pivotal step in the ongoing exploration of our Solar System's outer regions, building upon the legacy of missions like Cassini, Galileo, and Juno. It is designed to address two of ESA's Cosmic Vision's key scientific themes: understanding the conditions for planet formation and the emergence of life, and investigating the workings of the Solar System. By providing detailed comparative characterizations of the Galilean moons and their potential habitability, JUICE will enhance our understanding of giant planet systems, offering a paradigm for exploring similar systems throughout the galaxy.

Currently, the spacecraft is in its cruise mode along the Solar System, performing some gravity assists to arrive at the Jovian system in July 2031. During this phase, the spacecraft performs four orbits around the

[1]The Jovian system refers to the planet Jupiter and its surrounding environment, which includes its moons; Io, Europa, Ganymede and Callisto, the magnetic field, and radiation belts.

Sun, two Earth gravity assist, one Lunar-Earth gravity assist and one Venus gravity assist to be sure to take enough velocity to transfer to Jupiter in January 2029.

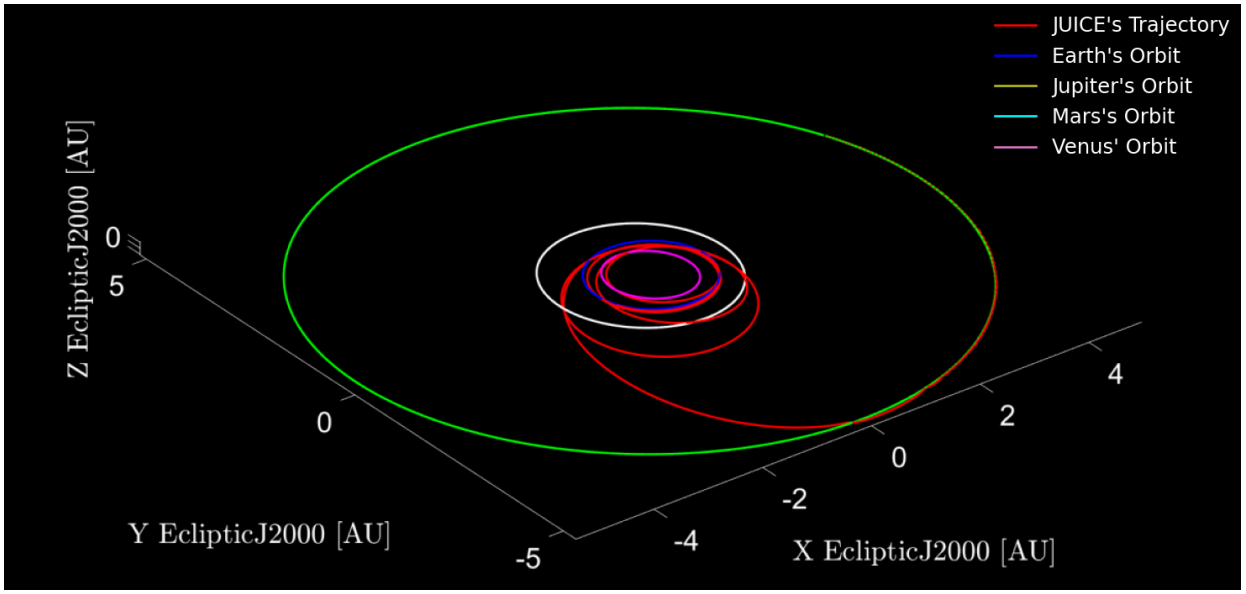


Figure 2.2: JUICE's journey along the Solar System. In red, can be observed JUICE's trajectory while in blue, yellow, cyan and purple the Earth's, Jupiter's, Mars' and Venus' orbits around the Sun, respectively. Axes are presented in Astronomical Units (AU) and Sun centered. EclipticJ2000 reference frame is used.

Once it gets enough velocity to go to Jupiter the spacecraft will start the different operation phases six months before its arrival [21]:

1. **Science mission begins:** JUICE will begin its scientific mission about six months before entering orbit around Jupiter, conducting observations as it approaches its destination.

2. **Gravity assist flyby of Ganymede:** Once in the Jovian system, a gravity assist flyby of Jupiter's largest moon Ganymede, also the largest moon in the Solar System, will help the spacecraft enter orbit around the gas giant. While orbiting Jupiter, the spacecraft will spend four years making detailed observations of Jupiter and three of its largest moons: Ganymede, Callisto, and Europa.

3. **Flybys of Europa:** During the tour, JUICE will make two flybys of Europa in July 2032. Europa has strong evidence for an ocean of liquid water under its icy shell. JUICE will study the moon's active zones, surface composition, and geology. It will also search for liquid water under the surface and study the plasma environment around Europa. Additionally, JUICE will explore the moon's tiny atmosphere and hunt for plumes of water vapour.

4. **Sequence of Callisto flybys:** A sequence of twelve Callisto flybys, from August 2032 to July 2033, will study this ancient, cratered world that may also harbor a subsurface ocean and will change the angle of JUICE's orbit with respect to Jupiter's equator, making it possible to investigate the polar regions and environment of Jupiter at higher latitudes.

5. **Ganymede and Callisto flybys for orbit adjustment:** JUICE's orbit will be adjusted through a series of flybys of Ganymede and Callisto. This will properly orient the spacecraft while minimizing

the amount of propellant expended. As a result, JUICE will be able to enter orbit around Ganymede in December 2034, making it the first spacecraft to orbit another planet's moon. The initial elliptical orbit will be followed by a 5000 km-altitude circular orbit, and later a 500 km-altitude circular orbit.

6. **Investigation of Ganymede:** JUICE will investigate Ganymede's unique magnetosphere, which is the only one in the Solar System. It will explore the moon's internal magnetic field and its interaction with Jupiter's plasma environment. Additionally, JUICE will study Ganymede's atmosphere, surface, subsurface, interior, and internal ocean, considering the moon as a planetary object and a possible habitat..
7. **Orbit decay and mission end:** Over time, JUICE's orbit around Ganymede will naturally decay. Eventually, there will not be enough propellant to maintain it, and the spacecraft will make a grazing impact onto the surface at the end of 2035.

2.1.1 Radar for Icy Moons Exploration (RIME)

RIME [22] is a radar sounder specifically designed to penetrate the icy surfaces of Jupiter's moons Ganymede, Europa, and Callisto, reaching depths of up to 9 km. This capability allows the study of subsurface geology, geophysics, and the detection of potential subsurface water reservoirs.

RIME is a nadir-looking active radar instrument that transmits low-frequency radio waves. These waves penetrate the subsurface and reflect off layers and structures with different dielectric properties, creating depth images called radargrams.

The main scientific objectives of the instrument are:

- Studying the subsurface geology and geophysics of Ganymede, Europa, and Callisto.
- Detecting possible subsurface water reservoirs, both present and past.
- Mapping thermal structures and ice impurities to understand geological processes and material exchange.
- Constraining the ice shell thicknesses, ice-ocean interfaces, and the role of convective processes.
- Performing both exploratory observations and focused hypothesis tests based on radar soundings.

It is worth noting that for the purpose of this thesis, it will not be used the full instrument specifications but rather a simplification based on real data. Consequently, the size of the antenna footprint on the ground is given by [23]:

$$\rho_{alt} = h\theta_{3dB} \approx \frac{h\lambda}{L_a} \tag{2.1}$$

$$\rho_{act} = R_M(\pi - 2\theta_{act}) \tag{2.2}$$

where ρ_{alt} and ρ_{act} are the footprint sizes in the along-track and across-track directions respectively, h

is the orbit height, θ_{3dB} is the 3dB aperture of the antenna, R_M the radius of the specific moon and $\theta_{act} = \arcsin(R_M/(h + R_M))$ is the angle between the nadir direction and the tangent to the moon's surface passing through the orbiter position, as can be seen in Fig.2.3

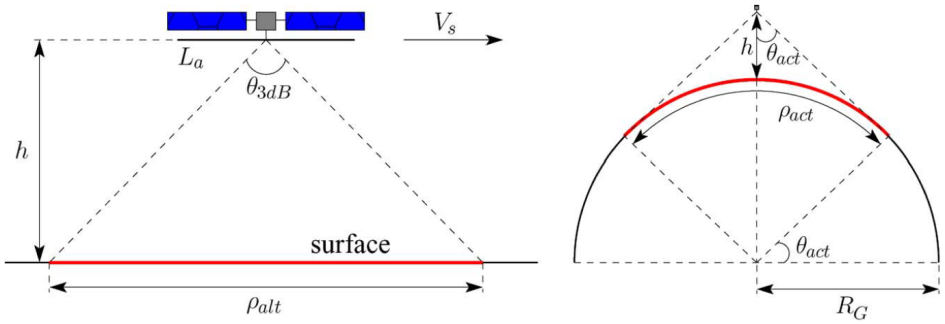


Figure 2.3: Observation geometry of sub-surface radar in the along-track and across-track directions. This figure has been taken from [23].

2.1.2 Imaging Sub-system JANUS

JANUS [24] is the narrow-angle camera onboard JUICE (see Fig. 2.4), covering visible wavelengths from near Ultraviolet (UV) to Near Infrared (NIR). The instrument is designed for detailed imaging of the Galilean moons, focusing on surface and subsurface interactions, geological features, and environmental processes.

JANUS will contribute to the study of icy satellites such as Ganymede, Europa, and Callisto by capturing high-resolution images to investigate tectonics, cryovolcanism, and impact craters. Imaging will help understand the geological evolution, surface ages, and dynamics of the moons' lithospheres.

Thus, the main scientific objectives of the instrument are the following:

- **Icy Moons:**
  - Study of tectonic, volcanic, and impact features on Ganymede, Europa, and Callisto.
  - Exploration of surface-subsurface exchanges, cryovolcanism, and lithospheric dynamics.
  - Mapping of surface composition using multi-filter imaging.
- **Jupiter's Atmosphere:**
  - Study tropospheric dynamics, cloud systems, and lightning activity.
  - Observe stratospheric disturbances and water meteorology.
  - Investigate aurora activity and polar hazes in the upper atmosphere.
- **Io's Volcanism:**
  - Monitor volcanic activity and changes in Io's surface.
  - Investigate the composition and transport of volcanic plumes.

To meet these objectives, JANUS has a resolution ranging from 3000 m/px to less than 10 m/px depending on the target and for both nadir and off-nadir pointing. In addition, its spectral range goes from 340 to 1080 nm covered by its 13 filters, allowing it to detect a large number of chemical compounds. Finally, the main optical features are:

FOV	1.29^{o} x 1.71^{o}
IFOV	15 μrad
Detector size	2000 x 1504 px
Pixel size	7 μm
Focal length	467 mm

Table 2.1: JANUS optical features. FOV refers to the angular extent of the observable area captured by the camera at any given moment. It defines the portion of the target body that can be imaged in a single exposure. IFOV refers to the angular size of a single pixel in the camera's sensor. It determines the camera's resolution by defining the smallest area on the target body that can be captured by each pixel. The data in the table have been extracted from [25].

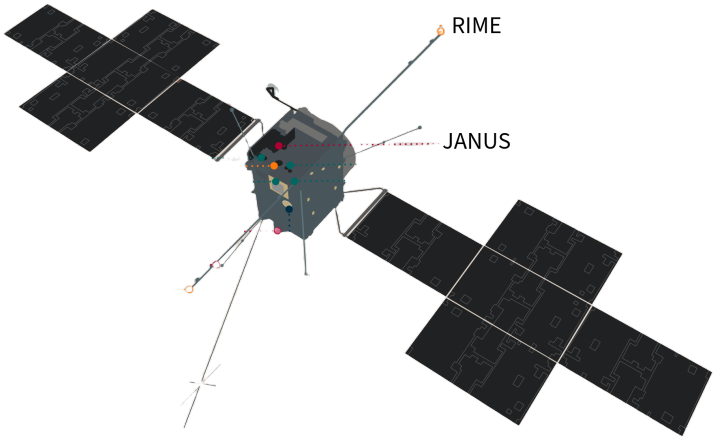


Figure 2.4: JUICE spacecraft body with indications of where RIME and JANUS are. This figure has been taken from [26].

Chapter 3

Methodology

In this chapter, the techniques employed for the development of this project are introduced, followed by the benchmarking carried out to demonstrate the performance.

3.1 Genetic Algorithms

The main key of this thesis is the GAs, which plays a crucial role in the future of scheduling optimization tools sharing space with other optimization algorithms, such as Particle Swarm (PS) or Tabu Search (TS).

Genetic algorithms are computational techniques inspired by the principles of biological evolution [27]. These algorithms serve as powerful search methods for solving a wide range of problems, while also providing models for simulating evolutionary processes. The fundamental concept involves representing potential solutions as binary strings in a computer's memory, similar to the genetic makeup of living organisms. Over successive iterations, these binary strings, or *individuals* are modified to simulate the process of natural selection, wherein the fittest solutions are more likely to survive and propagate.

In a genetic algorithm, a population of candidate solutions is initially generated at random. Each individual within this population represents a potential solution to a specific problem. Differences among individuals give rise to varying levels of *fitness* which are used to determine which individuals are more likely to survive and reproduce. The fitness of an individual is usually evaluated using a predefined function that quantifies how well the candidate solution performs in the given problem domain.

The evolution of the population is guided by the principles of *selection*, *crossover*, and *mutation*. During selection, individuals who demonstrate better fitness are more likely to be chosen to propagate to the next generation. For the correct evolution of the population, genetic operators that introduce variation are presented, such as crossover and mutation. These operators can be adapted to various types of variable representations, including real-values, binary strings, and integers [28]. Crossover is a recombination operator that exchanges segments between two parent solutions to generate new offspring, while mutation introduces random changes—ranging from bit flipping in binary strings to more complex modifications in real-valued or

integer-based representations. Through these operations, a new generation of candidate solutions is created, ideally with improved fitness over the previous generation.

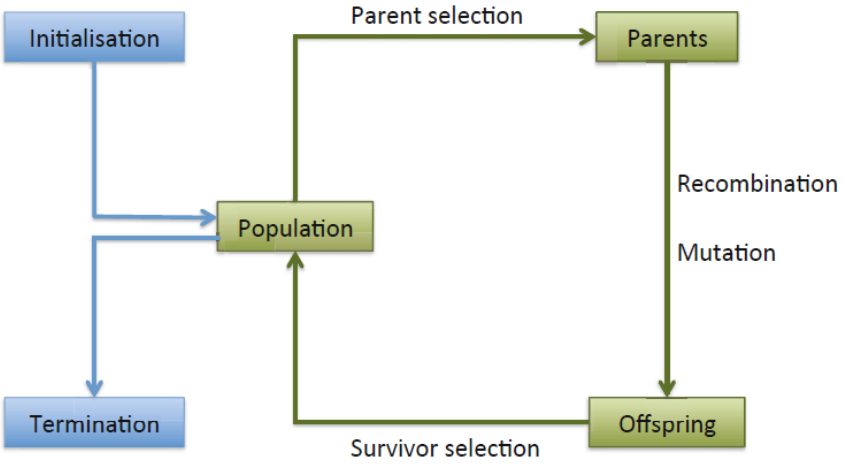


Figure 3.1: General flowchart of an Evolutionary Algorithm. This figure has been taken from [29].

This cycle (see Fig. 3.1) of evaluation, selection, and genetic variation is repeated over many generations. As the process unfolds, the population evolves, and the overall fitness of the individuals tends to improve, converging towards optimal or near-optimal solutions for the problem at hand.

The concept of genetic algorithms belongs to a broader category of optimization techniques known as Evolutionary Algorithms (EAs). Despite their variations, all EAs share a common foundation: given a population of candidate solutions, competition for limited resources drives natural selection, enhancing the fitness of the population over time. In these algorithms, the fitness of each candidate is assessed using a quality function that needs to be maximized or minimized. Higher fitness candidates are more likely to be selected for recombination and mutation, generating new offspring that undergo the same evaluation and selection process.

An important concept in genetic algorithms is the distinction between *genotype* and *phenotype* (Fig. 3.2), which is parallel to biological evolution. In biological systems, the genotype represents the underlying genetic code, while the phenotype is the physical manifestation of that genetic information. In the context of GAs, the genotype corresponds to the encoded candidate solution, whereas the phenotype is the actual solution derived from that encoding. Genetic operators such as crossover and mutation are applied to the genotypes, while the fitness of each candidate is determined by evaluating the corresponding phenotype.

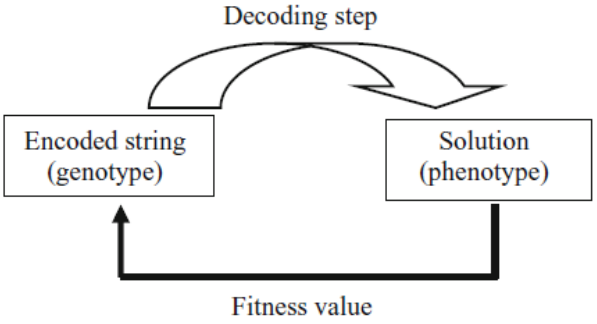


Figure 3.2: Genotype & Phenotype scheme. This figure has been taken from [30].

In mathematical terms, genetic algorithms can be characterized by a set of operations applied to a population of genotypes over discrete time steps. The population at a given time step is subjected to random variations, followed by a selection process that chooses individuals for mating based on their fitness. The final step involves replacement selection, where offspring compete with older individuals for inclusion in the next generation. This process gradually refines the population, favouring individuals with better fitness.

The effectiveness of genetic algorithms lies in their ability to explore and exploit the search space efficiently. The selection focuses on high-fitness solutions —or low, depending on the objectives—, while genetic operators introduce diversity, thereby preventing premature convergence on sub-optimal solutions. Crossover combines different parts of successful solutions (*exploitation*) and mutation introduces new possibilities (*exploration*) [31], allowing the algorithm to search a wider area of the solution space.

A key characteristic of genetic algorithms is their operation on encoded representations rather than on the solutions themselves. Before determining their fitness, these encoded solutions are translated into actual solutions for the problem domain. The fitness values of these solutions are then associated with their respective genotypes. This indirect method of working with solutions allows genetic algorithms to be applied to a broad range of complex problems, from optimization and game strategies to visual image generation and even the creation of computer programs.

Overall, genetic algorithms provide a flexible and robust method for solving complex optimization problems, leveraging principles from natural evolution to navigate vast search spaces and evolve high-quality solutions over successive generations.

3.2 Multi-Objective Optimization

As mentioned in the previous section, GAs are notable for their usefulness in optimising problems due to their ability to move within the solution space, finding those individuals that present better characteristics (genotype and phenotype) through their fitness.

Therefore, the single-objective should be easily extrapolated onto the multi-objective by making some changes in this fitness function [32]. The first approach is to adjust the fitness function to incorporate multiple functions using a weighted sum as follows,

$$f(x) = \sum_{i=1}^{k} w_i \cdot f_i(x) \tag{3.1}$$

where x is the domain of feasible solutions, w_i the weight for the i-th objective function and k the number of objectives. However, this is not the approach we will employ in this thesis.

In a single-objective optimization problem, it is typically seeked to minimize or maximize a single objective function $f(x) : R^d \to R$, where $x \in R^d$ represents a solution in the decision space. In this case, each solution x is mapped to a single scalar value $f(x) \in R$, representing its objective function value.

However, in a multi-objective optimization problem with N objectives, the aim is to optimize a set of objective functions simultaneously. When transitioning from single-objective optimization to multi-objective optimization with N objectives using the second approach, the solution space expands from R to R^N.

$$\mathbf{F}(x) = (f_1(x), f_2(x), \ldots, f_N(x)) : R^d \to R^N. \tag{3.2}$$

Here, each solution $x \in R^d$ is now mapped to an N-dimensional vector $\mathbf{F}(x) \in R^N$. The solution space is therefore represented in R^N, with each dimension corresponding to a different objective function. This expansion from a one-dimensional space R to an N-dimensional space R^N introduces several new challenges:

- **Selection of Individuals:** In single-objective optimization, the selection of individuals for reproduction is straightforward: we typically select those with the highest —or lowest— fitness, i.e., the best values of the objective function. In multi-objective optimization, however, there is no single "best" solution. Instead, solutions are often incomparable, as one solution may be better in some objectives while worse in others. This creates the need for alternative selection criteria, such as dominance-based approaches, to identify a set of non-dominated solutions that form what is known as the *Pareto Front*.
- **Exploration of the Solution Space:** As the number of objectives N increases, the solution space becomes more complex and expansive. Effective exploration of this high-dimensional space is critical to identifying diverse and high-quality solutions. However, simply focusing on a set of non-dominated solutions can lead to poor diversity and suboptimal exploration. Thus, additional strategies are required to ensure a deeper exploration of R^N.

3.2.1 Pareto Front

As mentioned before, in multi-objective optimization the individual selection is not as straightforward as for multi-objective, thereby, new methods appeared to ensure that the best individuals are selected to pass on to the next generations.

The method *Pareto Front* [33] consists of determining those individuals that are not dominated by others based on some criteria. Therefore, an individual belongs to a Pareto Front if it is not dominated by any other individual in the population. This means that

- There is no other individual that is better in all objectives.
- There is no other individual that is better in at least one objective and equal in the rest of them.

Then, based on these criteria, the algorithm that will allow the selection of the best individuals from the population is coded as follows:

Algorithm 1 Pareto Front Construction Algorithm

```
Initialize the maximum number of fronts, NFRONT ← 10
Initialize an empty front list for each individual, front ← [None] * getPopulationSize()
for f ← 0 to NFRONT − 1 do
    for i ← 0 to getPopulationSize() − 1 do
        if front[i] ≠ None then
            Continue
        end if
        if (fit[i][0] > 10) and (fit[i][1] < 1) then
            Continue
        end if
        Initialize dominant ← True
        for ii ← 0 to getPopulationSize() − 1 do
            if i = ii then
                Continue
            end if
            if (front[ii] ≠ None) and (front[ii] < f) then
                Continue
            end if
            Set better_in_all ← True if (fit[ii, k] ≤ fit[i, k])∀k ∈ [0, Ngoals)
            Initialize better_in_at_least_one ← False, n_equal ← 0, n_better ← 0
            for k ← 0 to Ngoals − 1 do
                if fit[ii, k] = fit[i, k] then
                    n_equal ← n_equal + 1
                else if fit[ii, k] < fit[i, k] then
                    n_better ← n_better + 1
                end if
            end for
            if (n_equal + n_better = Ngoals) and (n_better > 0) then
                better_in_at_least_one ← True
            else
                Continue
            end if
            if better_in_all or better_in_at_least_one then
                dominant ← False
                break
            end if
        end for
        if dominant then
            front[i] ← f
        end if
    end for
end for
for i ← 0 to getPopulationSize() − 1 do
    if front[i] = None then
        front[i] ← NFRONT + 1
    end if
end for
Set sorted ← False
```

In the following Figure 3.3, it is represented a set of individuals in a population where the Fitness function is defined as the Euclidean distance to each of the objectives points. Then, the individuals are generated along the y-axis from the middle point between the objectives, increasing their position in one unit of distance in the 2D Euclidean space. In this way, it will be appreciable how the Pareto Front algorithm works.

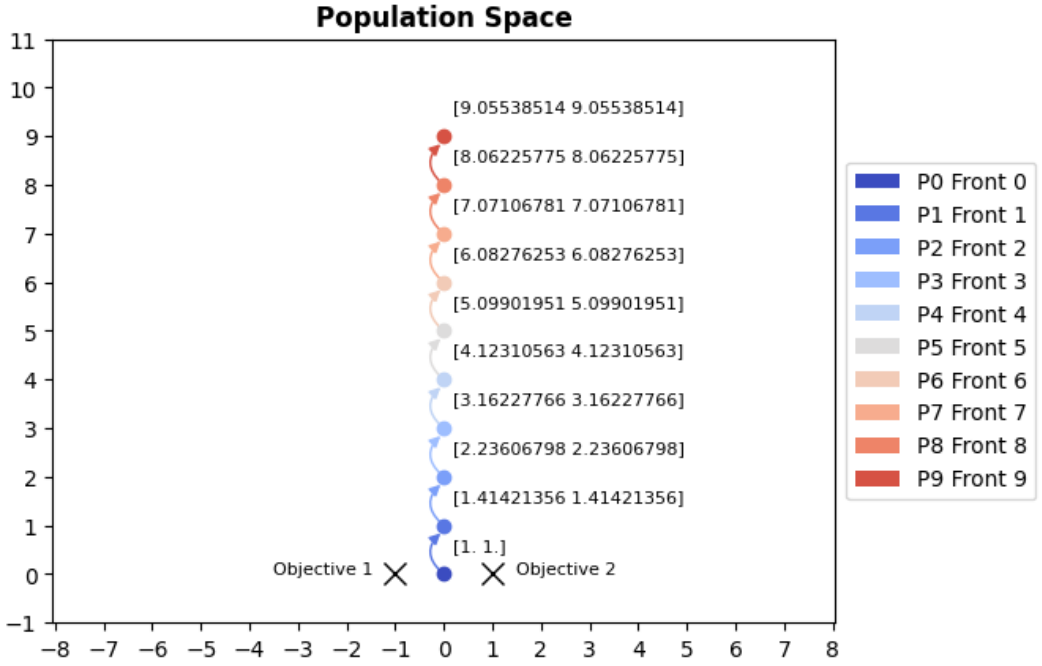


Figure 3.3: Pareto Front explanation in Population Space. The printed values close to the points are the *Fitness* values, i.e., the Euclidean distance to each one of the objective points.

It can be seen in Fig. 3.3 that Pareto Fronts are composed of those individuals that are non-dominated in the case of Front 0, or dominated only by individuals in the previous front for the following cases. Hence, as moving along the vertical axis, the individuals will belong to "worse" Pareto Fronts, since their Fitness is greater —further to objective points— than the previous ones. However, *what happens if there are two or more individuals who are better in one objective but worse in the other? How is it defined the non-dominance status in that case?*

To answer these questions, the Fig. 3.4 shows what happens when introducing a new individual that is closer to one of the objective points but further to the other than one individual in the first Pareto and one in the second Pareto.

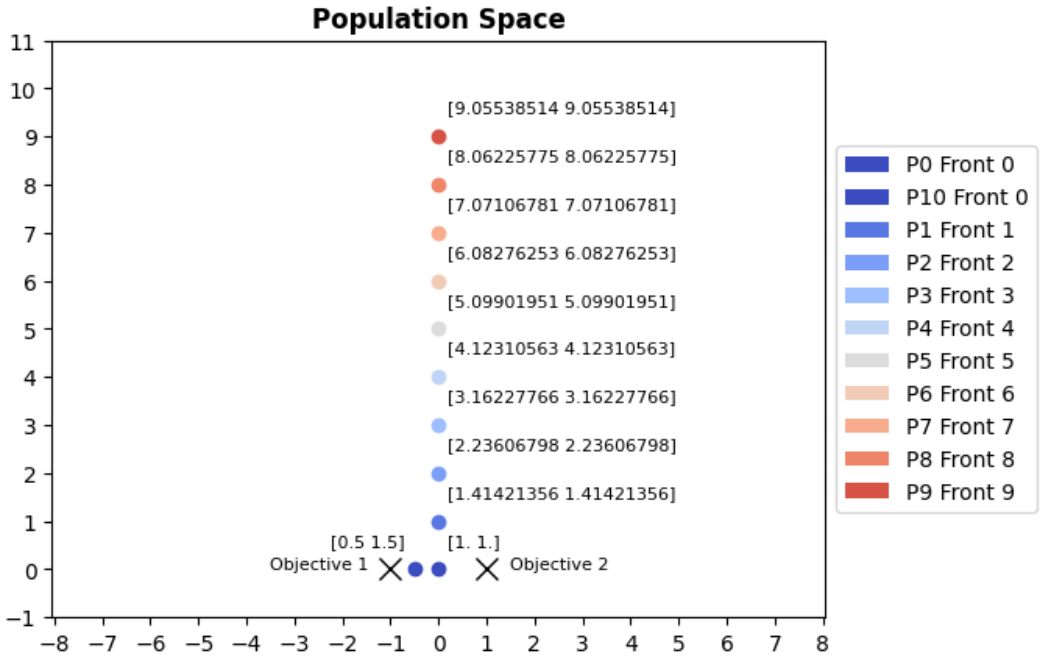


Figure 3.4: Pareto Front explanation from Population Space. In this case, an individual who performs better on one objective but worse on the other compared to both the first and second individuals is introduced.

It is shown that the new individual belongs to the first Pareto because it is not dominated by any other, even though it is worse than the individual from the second Pareto in one of the objectives. This means that being from a superior rank does not mean that it has to dominate every individual, but it is not dominated by any other individual in the population.

Therefore, the Pareto Front method ensures that the individuals from the first Pareto are the best individuals who meet all the objectives simultaneously, so there will be no individual who is better on all the objectives than any of those in the first Pareto.

3.2.2 Crowding Distance

As mentioned at the beginning of this section, one of the challenges that this multi-objective approach introduces is the exploration of the solution space, since the space has N dimensions, being N the number of objectives.

In this thesis, the method called *Crowding Distance* is implemented [34]. The crowding distance confidently serves as a robust measure that quantifies the proximity of an individual to its neighbours within the objective space. It calculates the distance from each solution to its nearest neighbours across the various objectives, effectively promoting diversity through its higher crowding distance since a higher crowding distance implies that a solution is in a less crowded area of the solution space, which is desirable for maintaining diversity. This crucial metric substantially contributes to the cultivation of a widely dispersed array of solutions along the Pareto front, thus ensuring the exploration of different areas within the solution space and enhancing the overall search process.

Therefore, this method presents significant advantages over just selecting the individuals randomly from the Pareto front, such as:

- Maintains Diversity: By favouring solutions with a higher crowding distance, the algorithm maintains a broad range of solutions.
- Improves Exploration: Solutions that are spaced apart are more likely to explore different trade-offs between objectives, leading to a more comprehensive understanding of the Pareto front.
- Prevents Dominance of Specific Regions: Without crowding distance, the algorithm might focus excessively on certain regions, potentially missing other valuable areas of the solution space and prematurely converging to sub-optimal regions.

To maintain maximum diversity in the population, an infinite crowding value is assigned to the individuals on the edges of the Pareto front. This will elevate them as the best individuals and enable exploration of the entire Pareto range, as it can be seen in the following algorithm.

Algorithm 2 Crowding Distance Calculation

Initialize the maximum number of fronts, $NFRONT \leftarrow 10$
Initialize the crowding distance list, crowd $\leftarrow [0] *$ getPopulationSize()
for $f \leftarrow 0$ to $NFRONT - 1$ **do**
    front_index $\leftarrow \{i \mid \text{front}[i] = f\}$ ▷ Indices of individuals in Pareto front f
    if |front_index| $= 0$ **then**
        Continue
    end if
    for $m \leftarrow 0$ to getNgoals() $- 1$ **do**
        Sort front_index by fitness values fit$[i][m]$ for objective m
        Assign infinite crowding distance: crowd[front_index[0]] $\leftarrow \infty$, crowd[front_index[-1]] $\leftarrow \infty$
        norm_range $\leftarrow$ fit[front_index[-1]][m] $-$ fit[front_index[0]][m]
        if norm_range $= 0$ **then**
            norm_range $\leftarrow 1$ ▷ Avoid division by zero
        end if
        for $j \leftarrow 1$ to |front_index| $- 2$ **do**
            next_fit $\leftarrow$ fit[front_index[$j + 1$]][m]
            prev_fit $\leftarrow$ fit[front_index[$j - 1$]][m]
            crowd[front_index[j]] $\leftarrow$ crowd[front_index[j]] $+ \left(\frac{\text{next\_fit} - \text{prev\_fit}}{\text{norm\_range}}\right)^2$
        end for
    end for
end for
Update crowding distances: crowd $\leftarrow \sqrt{\text{crowd}}$

To demonstrate the operation of the Pareto Front and Crowding Distance algorithms, a case has been designed in which there is a population of 500 individuals, whose Fitness function will be the Euclidean distance to two points located at [-1, 0] and [1, 0], as in Figs. 3.3 and 3.4. But this time the individuals will not be placed manually, but will be randomly generated in the whole space R^2.

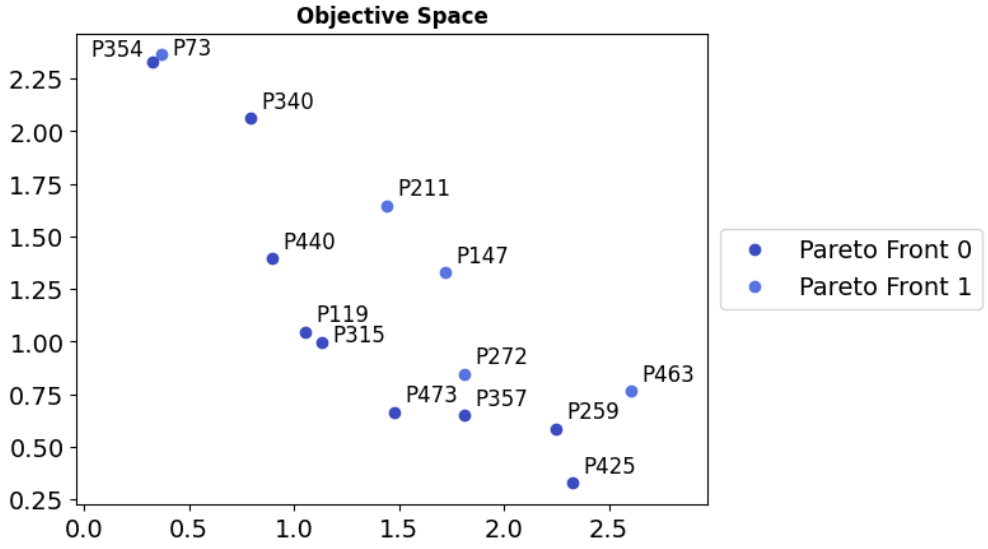


Figure 3.5: Non-sorted population after Pareto Front has been applied. The first two Pareto Front can be distinguished, thus fulfilling what has been seen previously, being all the points of Pareto 0 not dominated by any other point, while those of Pareto 1 are dominated by at least one point of the first Pareto.

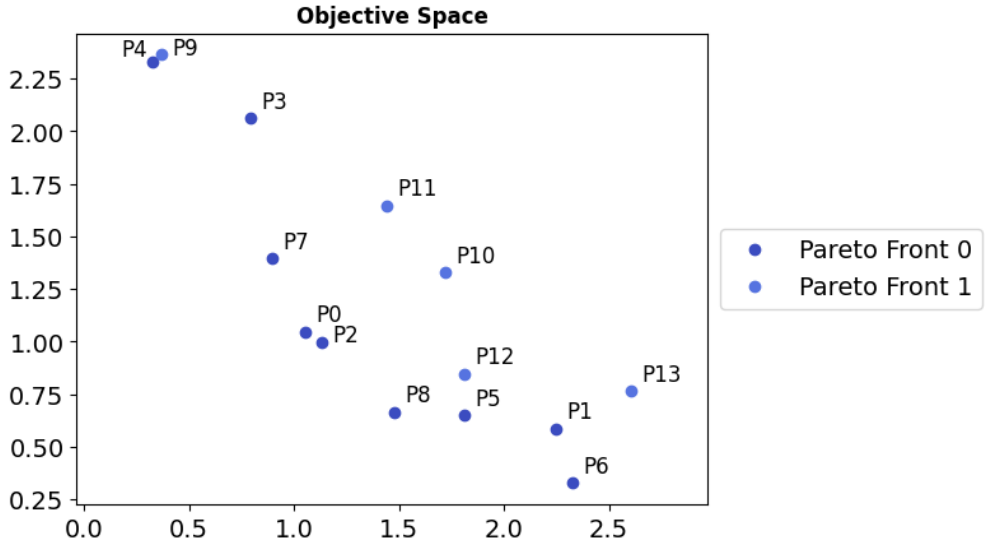


Figure 3.6: The population has been sorted after the Pareto Front algorithm is applied. Therefore, the individuals that have been selected to form the Pareto Fronts are placed in the first positions in the population, without any rule that takes into account the geometry in which they are located.

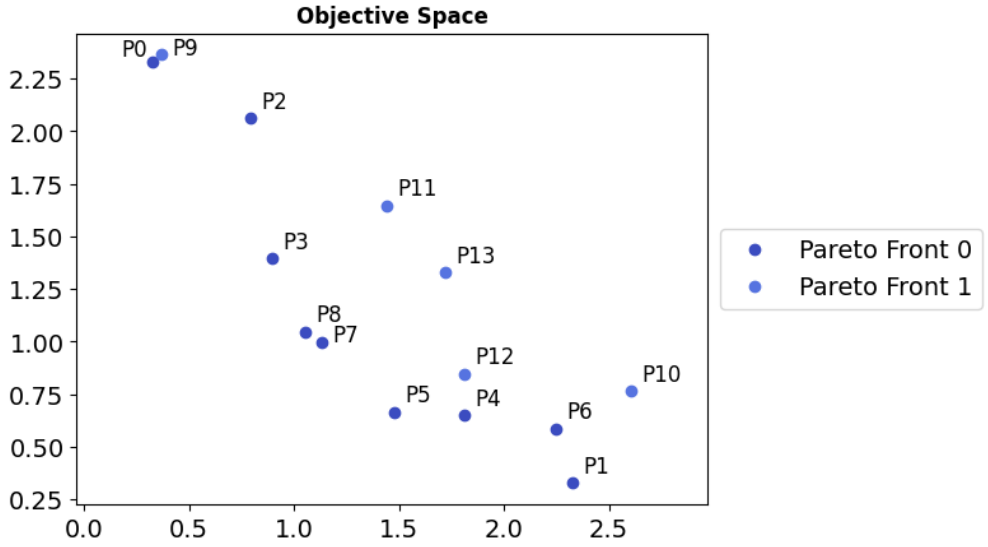


Figure 3.7: The population has been sorted using the Crowding Distance algorithm. Here, the individuals at the edges of the Pareto Fronts are placed first since they have been assigned an infinite crowding value. Then, those points that are in less crowded regions have priority over those that are closer to each other.

In the first figure (Fig. 3.5), the randomly generated population is displayed after applying the Pareto Front algorithm. The individuals are organized into two distinct Pareto fronts: Pareto 0, consisting of points not dominated by any others, and Pareto 1, whose individuals are dominated by at least one point from the first Pareto front. This establishes the hierarchy in the multi-objective optimization problem.

In the second figure (Fig. 3.6), it is observed that while the population is sorted based on the Pareto Front, there is no consideration for the geometric distribution of individuals within each front. Therefore, the points are arranged arbitrarily within their Pareto rank, potentially leading to clusters or uneven spread.

In contrast, the third figure applies the Crowding Distance algorithm, which prioritizes individuals located in less dense regions of the objective space. Points on the edges of the Pareto Fronts are prioritized due to their infinite crowding distance, ensuring that extreme solutions are preserved. Additionally, individuals in sparser regions are given priority over those in crowded areas, promoting diversity in the population by maintaining well-distributed solutions across the front. This method helps prevent premature convergence to local optima, as it avoids over-representation of solutions that are too similar and instead encourages exploration of the entire Pareto front.

3.3 Benchmarking

In this section, the algorithm's performance will be checked in some test problems, exploring different difficulties, and looking for the best combination of parameters to implement in the real problem.

3.3.1 Our Genetic Algorithm

The algorithm is designed in such a way that, through object-oriented programming, different classes communicate with each other, thus distributing the tasks and operating in a more effective way as it avoids duplication of data in the process.

These classes are:

- *basePMOT*: This class from the PMOT toolkit contains the different operators, such as reproduction, mutation and random function to initialize the population and the fitness function, in addition to the range of values for the inputs. Therefore, it is responsible for modifying the genotype of each individual within the population.
- *aga*: This class is in charge of the genetic algorithm configuration. It contains the proportion of mutants, elites, descendants, newcomers, etc. Furthermore, with this class, it is possible to repopulate the population. There is a wide number of parameters that can be configured, which makes it a very versatile and extensible class for any kind of problem.
- *amaga*: amaga is a class that inherits the aga class and its main function is the implementation of the multi-objective in the genetic algorithm. Therefore, this class contains the functions to create the Pareto fronts, apply the crowding distance, sort the individuals and see the state of the population, as well as complementary functions to visualize the results or calculate, for example, the metrics that will be used later to evaluate the algorithm.

With this information in hand, it is clear that the effectiveness of the algorithm will depend, not only on how the classes communicate between them but also on which parameters we select.

The number of elites, descendants, candidates for reproduction and newcomers is really important for the performance of the algorithm, in addition to the type of mutation and reproduction.

In this case, for the test problems the *BLX-*α reproduction function is implemented [35]:

$$x_d = x_i + \beta \cdot (x_j - x_i), \qquad \beta \in [-\alpha, 1 + \alpha] \tag{3.3}$$

where x_d is the offspring, $x_{i,j}$ are the parents and β is the recombination factor.

Concerning the mutation function, in this case, only a small change generated randomly from a uniform distribution with an adaptative σ is added, so as the population progress the change decreases.

Algorithm 3 Adaptive Mutation Function (`mutFun`)

1: **function** MUTFUN($fa = 0, ng = 0, scale\_0 = 0.2$) ▷ Adaptive mutation: Scale decreases as generations progress
2: scale $\leftarrow$ scale_0 $\times \left(1 - \frac{ng}{50}\right)$ ▷ Start with scale=0.2, reduce over generations
3: mutation $\leftarrow$ npy.random.normal(loc $= 0.0$, scale $=$ scale, size $=$ self.x.size)
4: self.x $\leftarrow$ self.x + mutation
5: **end function**

It is worth noting that for the test cases, the individuals have to accomplish certain values depending on the problem so that an extra correction method based on reflective conditions is applied. Then, if an individual overpasses the range it is automatically reflected to be within this interval.

3.3.2 Metrics

In this sub-section, the metrics that will be used to evaluate the performance are presented. Since we want to compare using Crowding Distance and simply sorting the individuals in the order they are selected for the Pareto front, we need to see the difference between the two methods, then we will use two metrics to measure the diversity of the first Pareto. All the metrics that we use are extracted from [36].

$$\text{Front Spread} = \sqrt{\sum_{i=1}^{n} \max(||a_i - b_i||)}, \qquad a, b \in F \tag{3.4}$$

where $i = 1...n$ are the number of objectives and a and b are individuals from the Pareto front F. This metric is called *Front Spread*, although we find it as M_3 in the mentioned document, and it represents the spread of the Pareto front, giving an insight into how much of the objective space is explored, so the higher the Front Spread, the better.

$$M_2 = \frac{1}{|F - 1|} \sum_{i,j}^{n} |||x_i - x_j|| > \sigma|, \qquad x_{i,j} \in F \tag{3.5}$$

where F is the Pareto front, $x_{i,j}$ are individuals from that front and σ is a specific threshold.

Therefore, this last metric M_2 evaluates how many individuals are separated from each other by more than a certain threshold. It gives an idea of how much the population evolves and converges to a local region. Thus, if this metric is high it means that there is more distance between individuals in the objective space.

Finally, although we are comparing the population diversity between the algorithms, it is also important to evaluate the accuracy.

$$\text{Convergence} = \frac{1}{F} \sum_{a \in F} \min(||a - x||), \qquad x \in X_P \tag{3.6}$$

where a are the individuals from Pareto F and x are the points that belong to the Pareto optimal X_p.

Thereby, this metric measures how close the individuals in the Pareto front are to the optimal solution. However, to use this metric we need to know the optimal solution, so it is not valid for every problem. Then,

as the metric measures the difference from the optimal solution, the lower the Convergence metric, the better.

3.3.3 Straight Line Test

Before comparing the algorithm with Crowding Distance and the algorithm without it, it is convenient to do some runs using a basic test to get the best parameters for the algorithm, as mentioned earlier, they are as important as the rest of the configuration.

For that, we will set a "straight line test". This consists of an easy problem in which the two objectives are two points in the 2-D Euclidean space, at [-1, 0] and [1, 0], so that the algorithm must fill the space between them, placing the individuals (points) in a straight line in the best of the cases, being the decision space the entire $\mathbf{R^2}$. Then, the Euclidean distance will be the *Fitness Function*.

For this first case, we will run the algorithm five times with the following parameters:

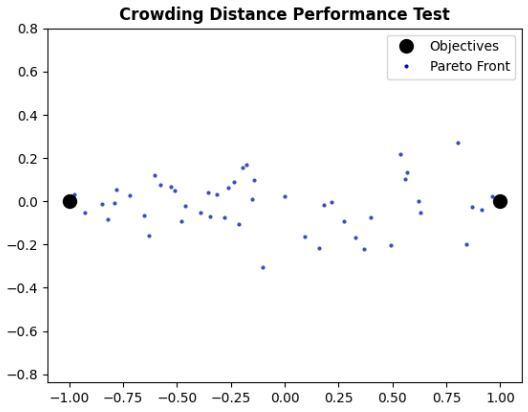
Combination	**1**	**2**	**3**	**4**	**5**	**6**
Population	200					
Elites	10%					
Newcomers	10%					
Mutants	80%	70%	55%	25%	10%	0%
Descendants	0%	10%	25%	55%	70%	80%
Can Mutate	15%					
Can Procreate	15%					
DOF	2					
Dimensions	2					

Table 3.1: Parameters Fine-Tuning. Different combinations are considered in order to give importance to both exploration and exploitation.

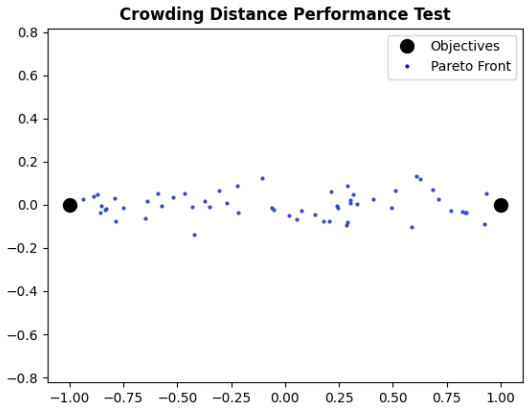
Therefore, we will implement the GA in this problem using the different settings presented in Table 3.1 to find the best combination for the following test cases.

Combination	**1**	**2**	**3**	**4**	**5**	**6**
Front size	50	60	77	135	163	180
Front Spread	1.9827	1.9892	1.9884	1.9849	1.9801	1.9094
M_2	17.04	20.80	23.83	44.21	47.23	43.08
Convergence	0.0145	0.0049	0.0041	0.0024	0.0025	0.0045

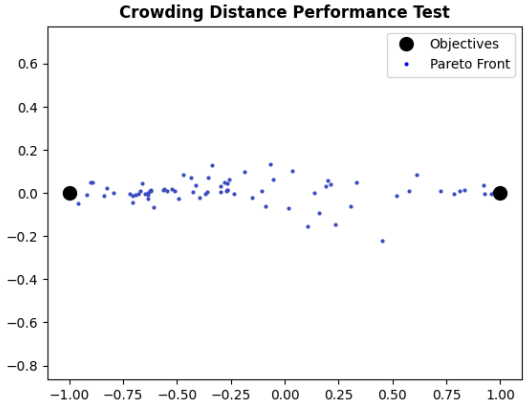
Table 3.2: Straight Line Test Results for parameter settings on Table 3.1. Both Front Spread and M_2 metrics evaluate the diversity in the population, while Convergence evaluates how close is the Pareto Front to the optimal one.



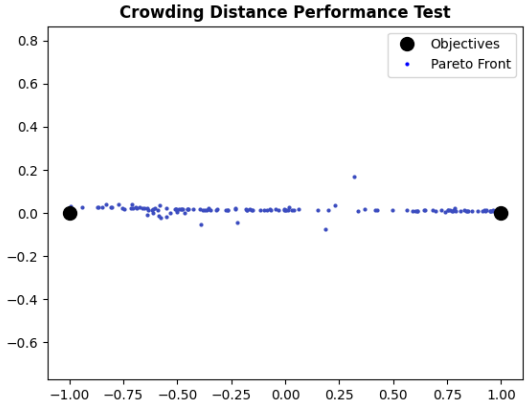
(a) Parameters setting 1



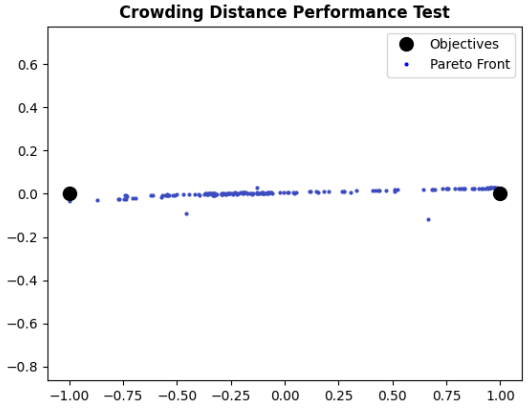
(b) Parameters setting 2



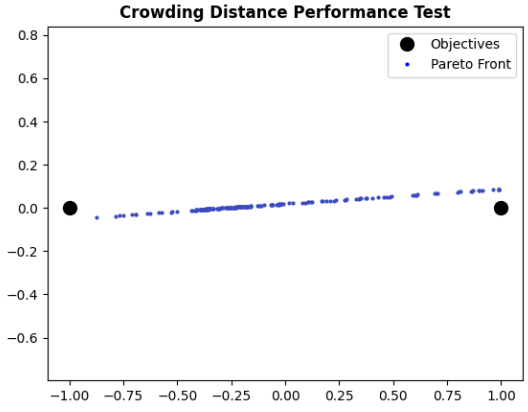
(c) Parameters setting 3



(d) Parameters setting 4



(e) Parameters setting 5



(f) Parameters setting 6

Figure 3.8: Straight Line Test with 6 different parameter settings, from exploration to exploitation.

It can be seen at a glance that the combinations of parameters 4, 5 and 6 from Table 3.1 are the ones closest to a straight line, corresponding to a lower number of mutants and, therefore, a higher number of descendants.

On the other hand, we observe in Table 3.2 that the values for the *Front Spread* metric are very similar among the different settings, except for the last one.

For the rest of the metrics, we appreciate that combinations 4 and 5 are the best ones. They present a higher diversity in their population but also a high accuracy in the results, achieving results really close to the optimal. Therefore, based on what we can see in Fig. 3.8 and in the previous table, we will use a half point between these two combinations from now, i.e., our population will have a 62% of descendants and 18% of mutants.

Therefore, the settings of our GA are presented in Table 3.3:

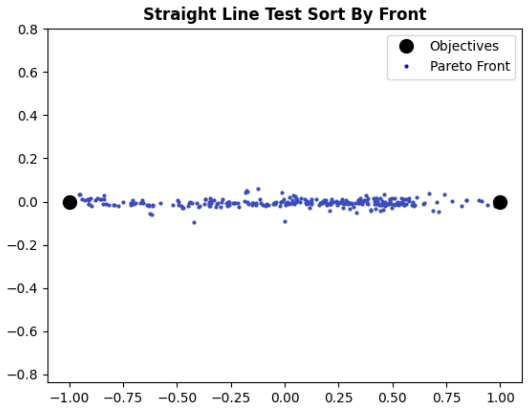
Not Crowding & Sort by Crowding Distance	
Population	400
Generations	40
Elites	40
Newcomers	40
Mutants	72
Descendants	248
Can Mutate	60
Can Procreate	60

Table 3.3: Final set of parameters for benchmarking problems. This is the combination of parameters that will be used in every test.

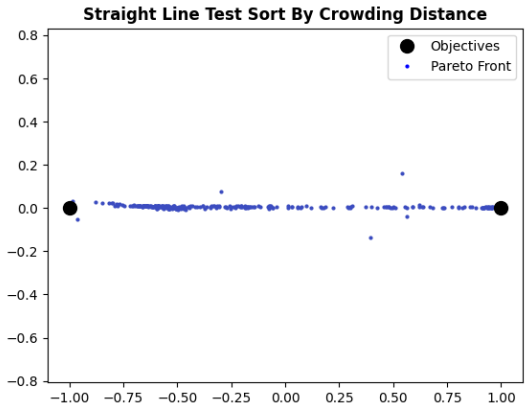
Then, starting from the same problem as we used to determine the best parameters, we run the algorithm implementing the Crowding Distance method and without it 3 times each.

Metric	**Not Crowding**	**By Crowding**
Front Size	322 ± 16	320 ± 10
M2	95.13 ± 0.49	96.90 ± 0.74
Front Spread	1.940 ± 0.044	1.995 ± 0.001
Convergence Metric	9.62e-4 ± 5.59e-6	9.63e-4 ± 7.19e-5

Table 3.4: Average metrics results with Standard Deviation for populations sorted by Front without using Crowding Distance and using the Crowding Distance algorithm in **Straight Line** test function.



(a) Straight Line test with parameters from Table 3.3 and population sorted by Pareto Front without Crowding.



(b) Straight Line test with parameters from Table 3.3 and population sorted by Crowding Distance.

Figure 3.9: Straight Line Test comparison between sorting by Front without Crowding Distance and sorting by Crowding Distance.

We can observe in Figs. 3.9 that the algorithm with *Crowding Distance* gets results closer to a straight line. On the other hand, if we take a look at Table 3.4 we see that the *Crowding Distance* algorithm preserves slightly better the diversity, giving greater Front Spread and M_2 metrics. Although both methods present a high-quality performance.

3.3.4 BIN Test

The BIN function is a test case originally proposed by Binh et al. [37] whose main purpose is to evaluate the performance of multi-objective evolutionary algorithms.

$$\begin{cases} f_1 = 4x_1^2 + 4x_2^2 \\ f_2 = (x_1 - 5)^2 + (x_2 - 5)^2 \end{cases} \tag{3.7}$$

where f_1 and f_2 are the objective functions and x_1 and x_2 are the parameters of each individual, i.e., the genotype and they must be within the range [-15, 30].

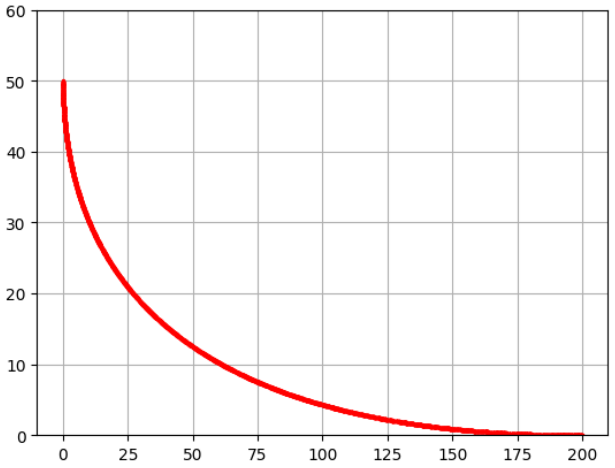


Figure 3.10: Pareto Optimal for the BIN function. This is the optimal Pareto Front that can be obtained when optimizing this function. The axis represents the two objective functions f_1 and f_2 in Eq. 3.7.

In Fig.3.10 we can observe that this is a 2-D problem, where Degrees of Freedom (DOF) = 2 and the Pareto optimal is a convex function.

Metric	By Front	By Crowding
Front Size	344 ± 8	343 ± 7
M2	121.50 ± 3.42	109.14 ± 3.65
Front Spread	14.94 ± 0.25	15.80 ± 0.01
Convergence Metric	0.024 ± 2.888e-4	0.021 ± 1.128e-3

Table 3.5: Average metrics results with Standard Deviation for populations sorted by Front without using Crowding Distance and using the Crowding Distance algorithm in **BIN** test function.

It is observable in Table 3.5 that both methods present good behaviour optimizing the BIN function. We can see that the diversity or *Front Spread* is higher in the *Crowding Distance* method, so it explores a greater range of the solutions space. However, the metric M_2 is smaller than for the Front sorting, so, in this case, this last method maintains a better distribution of the population along the Pareto Front.

3.3.5 FON Test

Finally, we will conclude this section with the FON function test. It was originally proposed by Fonseca et al. [38] and in this case, it is a 2-D function with DOF = 3.

$$\begin{cases} f_1 = 1 - \exp\left(-\sum_{i=1}^{3}\left(x_i - \frac{1}{\sqrt{3}}\right)^2\right) \\ f_2 = 1 - \exp\left(-\sum_{i=1}^{3}\left(x_i + \frac{1}{\sqrt{3}}\right)^2\right) \end{cases} \tag{3.8}$$

where $x_{1,2,3}$ are the DOF of the problem and f_1 and f_2 the objective functions. In this case, the variables to

optimize must be within the range

$$x_{1,2,3} \in [-4, 4]$$

.

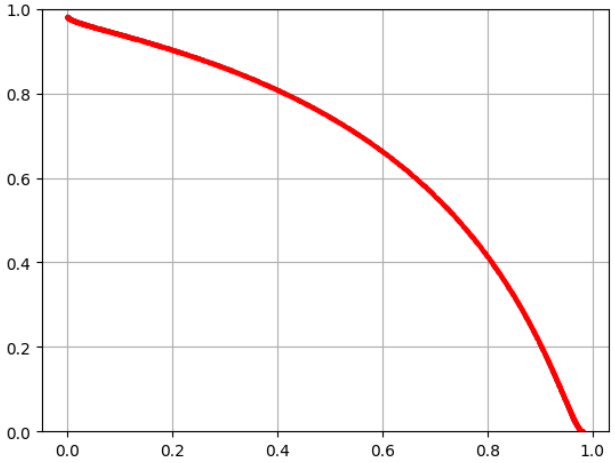


Figure 3.11: Pareto Optimal for the FON function. This is the optimal Pareto Front that can be obtained when optimizing this function. The axis represents the two objective functions f_1 and f_2 in Eq. 3.8.

As shown in Fig. 3.11, the Pareto optimal of this function is non-convex, which makes it inherently more challenging for GAs to explore and converge on the optimal solutions. In non-convex Pareto fronts, the solutions may not lie on a straight line or surface, meaning that crossover and mutation operators in GAs can struggle to generate offspring that lie near the Pareto-optimal front.

Furthermore, it has 3 DOF, meaning it has more variables (dimensions) to optimize compared to the BIN function, which has 2 degrees of freedom. The more DOF a function has, the larger and more complex the search space becomes. With more variables, the algorithm needs to explore a larger number of potential solutions and manage more complex trade-offs between objectives.

Comparison of Metrics Sorting by Front and Crowding Distance		
Metric	**By Front**	**By Crowding**
Front Size	292 ± 3	302 ± 6
M2	57.31 ± 11.40	66.84 ± 10.19
Front Spread	1.39 ± 1.56e-3	1.40 ± 3.31e-4
Convergence Metric	4.39e-4 ± 1.03e-4	2.76e-4 ± 5.21e-6

Table 3.6: Average metrics results with Standard Deviation for populations sorted by Front without using Crowding Distance and using the Crowding Distance algorithm in **FON** test function.

Table 3.6 shows how the *Crowding Distance* algorithm presents better performance, being better in the four metrics.

Therefore, although the differences between both methods for all the tests are small, we can conclude that

Crowding Distance generally performs better than the other way of sorting and preserves the diversity, indicating that it explores a wide range of the solution space, which can be of great use in the scheduling problem due to the nature of the data.

Chapter 4

Scheduling Problem

Here, we want to demonstrate the feasibility of using GAs to optimize the schedule of observations in remote sensing missions. Therefore, during this section, the problem formulation will be presented.

Before introducing the different aspects of the problem, it is worth noting that the code is divided into two different parts. On the one side, the first part involves the Genetic Algorithm and the operators to make the population evolve, so in this group are the *amaga* and *aga* classes mentioned in the previous section, and the *oPlan* class, which contains the operators adapted to the problem and some functions to make the calculations. On the other hand, the second part consists of the different ROI and the technical constraints and requirements of the problem, which is in charge of the computation of the data for feasible observations and the communication between classes to obtain better efficiency during the execution. In this group can be found the classes: *Instrument*, *ROIDataBase*, *roi*, *oPlanRoi* and *DataManager*.

4.1 Opportunity Windows

The first step to face this problem is to calculate the feasible opportunity time windows to operate the instruments, making observations in case of JANUS and scanning in case of RIME.

For that, some input information has been given to the code to start with the calculation. This input consists of a list of ROI given in a .txt file so that the *ROIDataBase* class reads, cleans and saves the information in a list of Python dictionaries, which will contain the following:

- **ROI_key**: it is an identification for each ROI that has been defined by ESA. Here is an example "JUICE_ROI_CAL_1_0_01", where CAL would be the moon where the region is located, in this case Callisto, and the number in format X_X_XX indicates the kind of region.
- **Body**: it is the body that is being studied, i.e., the target body.
- **Vertices**: the vertices of the ROI, which are saved in latitudinal coordinates.

After the previous data extraction, the *oPlanRoi* class is initialized for each ROI and saved in a Python list.

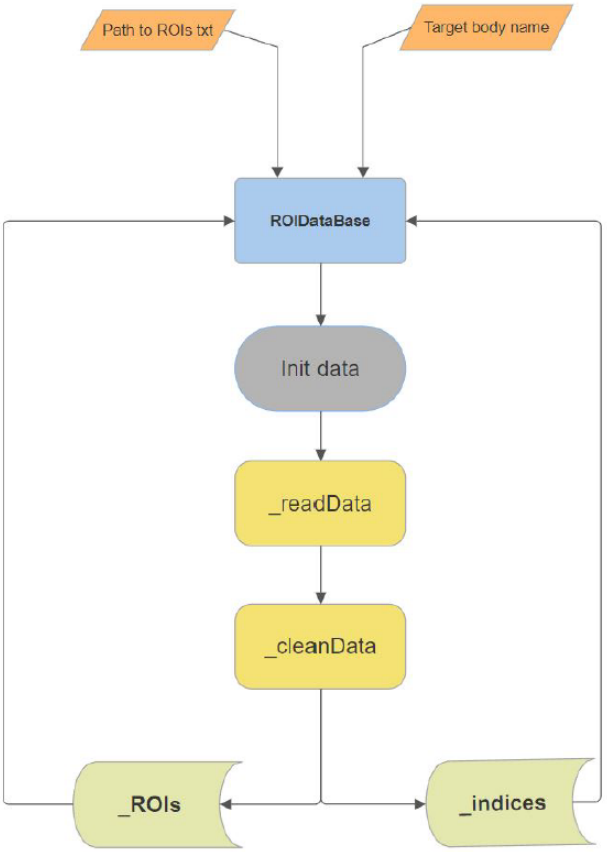


Figure 4.1: Scheme of how *ROIDataBase* class works. This figure has been extracted from [39].

This last class inherits the *roi* class, which computes the geometry of each region, saving the vertices in a new reference, the centroid and the area, values that will be for further use.

Then, the input data of the problem is introduced. In this case, the problem to optimize will be the schedule of observations and scanning of JUICE using JANUS and RIME during the 12 flybys it performs around Callisto from August 2032 to July 2033. Therefore, the solutions space is defined as the time window in which the different instruments take the best measurements in an organized way, taking into account the geometry constraints for each instrument based on their requirements and specifications.

Then, the search space will be constrained to the following dates in Table 4.1:

Flyby	Begin	End
1	2032 AUG 14 10:09:17	2032 AUG 15 10:09:17
2	2032 SEP 10 11:52:29	2032 SEP 11 11:52:29
3	2032 SEP 27 04:26:36	2032 SEP 28 04:26:36
4	2032 OCT 13 20:53:36	2032 OCT 14 20:53:36
5	2032 OCT 30 14:21:22	2032 OCT 31 14:21:22
6	2032 NOV 16 05:53:16	2032 NOV 17 05:53:16
7	2032 DEC 27 21:06:34	2032 DEC 28 21:06:34
8	2033 JAN 13 13:32:12	2033 JAN 14 13:32:12
9	2033 MAR 04 14:44:09	2033 MAR 5 14:44:09
10	2033 APR 06 23:29:30	2033 APR 07 23:29:30
11	2033 MAY 10 08:21:32	2033 MAY 11 08:21:32
12	2033 JUL 16 02:11:44	2033 JUL 17 02:11:44

Table 4.1: Official dates for JUICE flybys around Callisto. Data extracted from [40].

Then, as has been commented before, geometric and spacecraft attitude constraints will be applied to these time intervals to satisfy the mission requirements. To compute these limitations, the Python Science Opportunity Analysis (PSOA) toolbox, where most of the functions were coded by Paula Betriu, has been implemented.

The user specifies which constraints to consider and the acceptable range for each. This information is then passed to a function called *my_tw_finder* from the library coded by professor Manel Soria, *pySPICElib*, which applies the Bolzano theorem to determine which time intervals meet the defined constraints within the initial search space. The result is a set of refined time windows, one for each ROI, each potentially containing multiple intervals or none at all. These compliant intervals are always a subset of the original search space.

This results in two key outcomes:

- Multiple observation opportunities may exist within a single initial interval, particularly when the time frame is large. However, for the JUICE mission, due to the brevity of the flybys, it is more common to find only one or no valid intervals per flyby.
- The original, unconstrained time intervals are split into smaller, compliant intervals, reducing the general search space and creating a subdomain of acceptable time windows.

Once this process is completed, the original concept of the search space, whether it's defined by flybys or orbits, becomes irrelevant. Only the compliant time windows remain, giving greater flexibility to the genetic algorithm. This allows observation opportunities to shift between different flybys, provided a valid time window exists, facilitating the search for the optimal observation schedule.

4.1.1 PSOA toolbox

Some of the functions from the PSOA toolbox that has been employed in the frame of this project are presented in this sub-section.

Altitude

This function calculates the altitude of an observer above the surface of a target body at a specific point in time or over a range of time values as follows:

1. The target body is modelled as an ellipsoid.
2. For each time step, the sub-observer point on the surface of the target is computed.
3. The distance between the observer and the sub-observer point is calculated, which corresponds to the altitude.
4. The result is returned as either a scalar or a vector depending on the input type.

Algorithm 4 Altitude

Input:
  obs: String name of the observer body
  target: String name of the target body
  t: Time epoch(s) in TDB seconds past J2000 (can be a scalar or vector)
Output:
  alt: Observer altitude over the body surface at each time step in kilometers
Initialize `method` to 'INTERCEPT/ELLIPSOID' (modeling target as ellipsoid)
Set `abcorr` to 'NONE' (no aberration correction)
Retrieve the reference frame `tframe` of the target body using SPICE call
Ensure `t` is an array of time values using `np.atleast_1d(t)`
Initialize an array `alt` of zeros with shape corresponding to `t`
for `each time step` t_i `in` t **do**
    Compute sub-observer point on the target body surface using SPICE call
    Compute the observer position vector relative to the target body
    Calculate the altitude as the Euclidean norm of the vector difference between the observer and the sub-observer point
    Store the altitude in `alt[i]`
end for
if `t` is a scalar or contains only one value **then**
    Return `alt[0]` as a scalar
else
    Return `alt` as a vector
end if

Emission Angle

This function calculates the emission angle at a specific point on the surface of a target body. The emission angle is the angle between the normal to the surface at a given point and the line of sight vector from that point to an observer (such as a spacecraft). The methodology is:

1. The function first checks whether the surface point is given in latitudinal or Cartesian coordinates and converts it to Cartesian if necessary.

2. The surface normal vector is calculated using the SPICE function srfnrm based on an ellipsoidal model of the target.

3. The vector from the surface point to the observer is calculated.

4. The emission angle is computed as the angle between the surface normal vector and the observer's vector using the vector separation function.

5. The result is returned as an angle in degrees.

Algorithm 5 emissionang

Input:
  `srfpoint`: Surface point on the target body (in latitudinal or Cartesian coordinates)
  `t`: Time epoch(s) in TDB seconds past J2000 (scalar or vector)
  `target`: String name of the target body
  `obs`: String name of the observer body
Output:
  `angle`: Emission angle between the surface normal and observer's line of sight (in degrees)
Set `method` to 'ELLIPSOID' (modeling the target as a tri-axial ellipsoid)
Retrieve the target reference frame `targetframe` using SPICE call
Compute observer position vector `obsvec` using SPICE function `trgobsvec(srfpoint, t, target, obs)`
if `srfpoint` is in latitudinal coordinates **then**
    Convert latitudinal coordinates to rectangular (Cartesian) using SPICE functions
else
    Reshape `srfpoint` as a 3D vector
end if
if `t` is a vector of time values **then**
    for `each time step` t_i `in` t **do**
        Compute the surface normal vector `nrmvec` using `spice.srfnrm(method, target, t[i], targetframe, srfpoint)`
        Compute the emission angle `angle[i]` as the vector separation between `obsvec[i]` and `nrmvec[i]`
        Convert `angle[i]` from radians to degrees
    end for
else
    Compute the surface normal vector `nrmvec` for a single time step
    Compute the emission angle as the vector separation between `obsvec` and `nrmvec`
    Convert the angle to degrees
end if
Return the emission angle (as a scalar or array, depending on input time)

Ground-Track

This function computes the ground track of a spacecraft (observer) on the surface of a target body, such as a moon or planet, at a given time (or times). The ground track represents the sub-observer point, which is the location on the surface directly underneath the observer at each time step.

1. The function uses an ellipsoid model to represent the target body and calculates the sub-observer point based on this model.

2. For each time step, the sub-observer point is computed, and its Cartesian coordinates are converted into latitudinal coordinates (longitude and latitude).

3. The latitudinal coordinates are initially computed in radians and are converted into degrees.

4. The function handles both single time points and arrays of time points by checking the size of the input time array and adjusting the computation accordingly.

Algorithm 6 groundtrack

Input:
  `obs`: String name of the observer body (e.g., spacecraft)
  `t`: Time(s) in TDB seconds past J2000 epoch (can be a scalar or vector)
  `target`: String name of the target body
Output:
  `gtlon`: Longitude of the sub-observer point (in degrees)
  `gtlat`: Latitude of the sub-observer point (in degrees)
Set `method` to 'INTERCEPT/ELLIPSOID' (ellipsoid model for target body)
Set `abcorr` to 'NONE' (no aberration correction)
Obtain the body-fixed frame of the target body from SPICE as `tframe`
if `t` contains multiple values **then**
  Initialize arrays `sctrack`, `gtlon`, and `gtlat` for storage
  for `each time step` t_i **do**
    Compute the sub-spacecraft point using `spice.subpnt()`
    Convert the sub-spacecraft point to latitudinal coordinates (longitude, latitude) using `spice.reclat()`
  end for
else
  Compute the sub-spacecraft point at the single time step using `spice.subpnt()`
  Convert the sub-spacecraft point to latitudinal coordinates (longitude, latitude) using `spice.reclat()`
end if
Convert `gtlon` and `gtlat` from radians to degrees using SPICE's `spice.dpr()` function
Return the longitude and latitude of the ground track (`gtlon` and `gtlat`)

Phase Angle

The *phaseang* function calculates the phase angle at a given surface point on a target body, which is the angle between the vector pointing towards the illumination source (typically the Sun) and the vector pointing towards the observer (such as a spacecraft). This angle helps determine the relative illumination conditions at the surface point, which is useful in analyzing images and measurements taken from space.

1. If the surface point is given in latitudinal coordinates, the function converts it to Cartesian coordinates.

2. The position vector from the surface point to the observer is computed (trgobsvec function from PSOA).

3. The position vector from the surface point to the illumination source (the Sun) is computed (trgillvec function from PSOA).

4. The angle between the two vectors (observer and illumination) is computed using SPICE's vsep function, which returns the angular separation between two vectors. The angle is then converted to degrees.

5. If multiple time values are provided, the function calculates the phase angle for each time step.

Algorithm 7 phaseang

Input:
  `srfpoint`: Surface point on the target body (latitudinal or Cartesian coordinates)
  `t`: Time(s) in TDB seconds past J2000 epoch (can be a scalar or vector)
  `target`: String name of the target body
  `obs`: String name of the observer body
Output:
  `angle`: Phase angle (in degrees) between the illumination source vector and the observer vector
if `srfpoint` is in latitudinal coordinates **then**
    Convert `srfpoint` from degrees to radians
    Convert latitudinal coordinates to Cartesian coordinates using `spice.srfrec()`
end if
Compute the observer's position vector relative to the surface point using `trgobsvec()`
Compute the illumination source's position vector relative to the surface point using `trgillvec()`
if `t` contains multiple values **then**
    Initialize an array `angle` to store the phase angle for each time step
    for `each time step` t_i **do**
        Compute the angular separation between the observer vector and illumination vector using `spice.vsep()`
        Convert the angle from radians to degrees
    end for
else
    Compute the phase angle for the single time step using `spice.vsep()`
    Convert the angle to degrees
end if
Return the phase angle (as a scalar or array, depending on input time)

Illumination Zenith Angle

The *illzenithang* function calculates the illumination zenith angle, which is the angle between the normal vector to the surface at a given point on a target body and the vector pointing to the illumination source (typically the Sun). This angle is equivalent to the emission angle but with the Sun as the "observer." It helps determine the lighting conditions at the surface point.

Point in Polygon

The *pointinpoly* function checks if a given point (defined by longitude and latitude) is inside a specified polygon, using the Ray-Tracing Algorithm. This algorithm works by drawing a "ray" from the point horizontally and counting how many times the ray intersects the edges of the polygon. If the number of intersections is odd, the point is inside the polygon; otherwise, it is outside.

1. The longitude of the point is adjusted based on a specific coordinate system (shifting values outside the range of [-90, 90] degrees for longitude).
2. The function iterates through each edge of the polygon, checking if a horizontal ray extending from the point intersects the edge. If the number of intersections is odd, the point is inside the polygon.
3. The function computes the intersection of the horizontal ray and each polygon edge, adjusting the state of the inside variable as necessary.
4. The function returns True if the point is inside the polygon and False if it is outside.

Algorithm 8 pointinpoly

Input:
  `point`: Coordinates of the point [longitude, latitude]
  `polygon`: List of vertex coordinates defining the polygon
Output:
  `inside`: Boolean value indicating if the point is inside the polygon
Extract `x` (longitude) and `y` (latitude) from `point`
if `x` $\leq$ -90 **then**
    Adjust `x` by adding 180
else if `x` $>$ 90 **then**
    Adjust `x` by subtracting 180
end if
Initialize `inside` as `False`
Set `n` as the number of vertices in the `polygon`
Set `p1x`, `p1y` as the first vertex of the `polygon`
for each edge i from 0 to `n` **do**
    Set `p2x`, `p2y` as the next vertex of the polygon (i modulo n ensures a loop back to the first vertex)
    if `y` is between `p1y` and `p2y` **then**
        if `x` is less than or equal to `max(p1x, p2x)` **then**
            if `p1y` $\neq$ `p2y` **then**
                Compute the intersection `xinters` of the ray with the polygon edge:

$$\texttt{xinters} = \frac{(\texttt{y} - \texttt{p1y})(\texttt{p2x} - \texttt{p1x})}{\texttt{p2y} - \texttt{p1y}} + \texttt{p1x}$$

            end if
            if `p1x` = `p2x` or `x` $\leq$ `xinters` **then**
                Toggle the value of `inside` (`inside` = not `inside`)
            end if
        end if
    end if
    Set `p1x`, `p1y` to `p2x`, `p2y`
end for
Return `inside`

3-Points Angle

The *compute_3points_angle* function calculates the angle between two vectors defined by three points: point1, center, and point2. These vectors originate from a common center body and extend to point1 and point2, which are celestial bodies or spacecraft. The function computes the angle between these two vectors at a specific time using SPICE library functions.

1. The positions of point1 and point2 relative to center at a given time et are computed using SPICE's spkpos function.
2. Both vectors are normalized using SPICE's vhat function to obtain unit vectors.
3. The cosine of the angle between the two unit vectors is calculated using the dot product.
4. The angle is computed using the arccosine of the dot product and then converted to degrees.

Algorithm 9 compute_3points_angle

Input:
`et`: Ephemeris time in TDB seconds past J2000 epoch
`point1`: First point in SPICE nomenclature
`center`: Common center body in SPICE nomenclature
`point2`: Second point in SPICE nomenclature
Output:
`angle_degrees`: Angle in degrees between vectors from `center` to `point1` and `center` to `point2`
Set the reference frame `frame` to 'J2000'
Use SPICE function `spkpos` to get the position vector `pos1` of `point1` relative to `center` at time `et`
Use SPICE function `spkpos` to get the position vector `pos2` of `point2` relative to `center` at time `et`
Normalize `pos1` to get `unit_vector1` using SPICE's `vhat` function
Normalize `pos2` to get `unit_vector2` using SPICE's `vhat` function
Compute the dot product of `unit_vector1` and `unit_vector2` to get `cosine_angle` using SPICE's `vdot` function
Compute the angle in radians using the arccosine of `cosine_angle`:

$$\texttt{angle\_radians} = \arccos(\texttt{cosine\_angle})$$

Convert the angle from radians to degrees:

$$\texttt{angle\_degrees} = \text{degrees}(\texttt{abs}(\texttt{angle\_radians}))$$

Return `angle_degrees`

4.1.2 JANUS Opportunity Windows

As described at the beginning of this section, geometric and attitude constraints must be introduced to satisfy the scientific mission requirements and obtain the best results. In the case of the instrument JANUS, the result or objective to optimize is the resolution of the images that will be taken, which is computed as follows for each ROI in each time step:

$$R = \frac{\text{ifov} \times d}{\sqrt{\sin(90^\circ - \theta)}} \tag{4.1}$$

$$R_{\max} = \frac{\text{ifov} \times d_{\max}}{\sqrt{\sin(2^\circ)}} \tag{4.2}$$

where R is the resolution of the ROI, which is an approximation since it is the resolution obtained when pointing at the centroid of the region, ifov is the angular resolution of the instrument, d the distance between the observer and the surface point (region's centroid) and θ is the emission angle computed with the function *emissionang* that has been shown previously.

Regarding $R_{\max}$, it is the maximum limited resolution to be captured and it is defined to avoid possible singularities since the maximum emission angle that will be considered is 90^{o} and then the denominator will be zero. In the case of introducing a tuple of times instead of single values, a maximum distance is also defined, however, if time is a single value the maximum distance will be just the distance at this moment computed by *trgobsvec* function from PSOA.

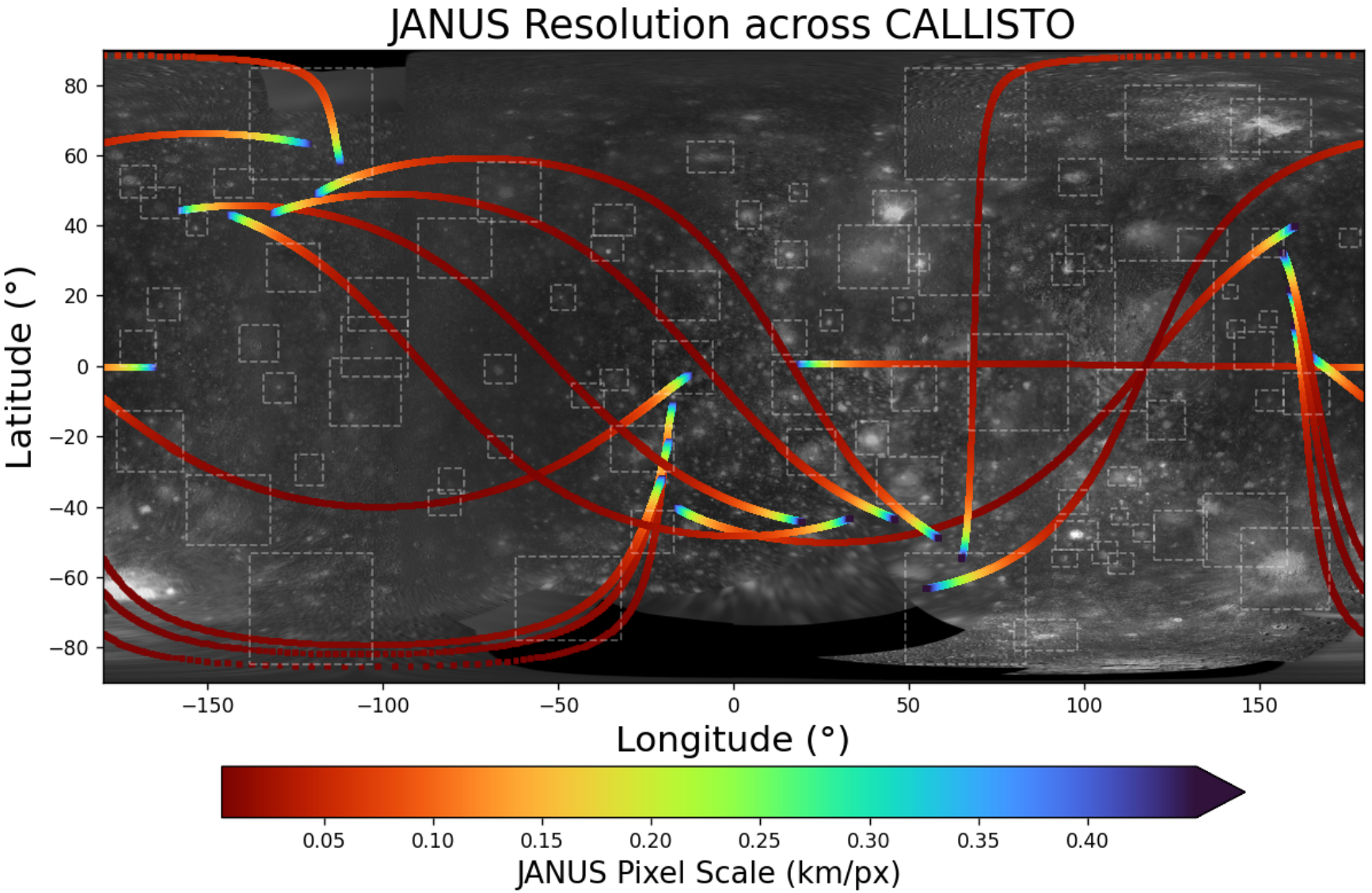


Figure 4.2: This figure shows the magnitude of the problem during JANUS observations. The resolution is computed with respect to the ground-track since the number of ROI is high, giving an idea of how many observation possibilities are during the different Callisto flybys. The resolution has been constrained to an altitude of less than 30000 km for better contrast in the color scale.

Therefore, with this context in mind, the constraints for the instrument JANUS are presented in Table 4.2:

Parameter	Max Value
Phase Angle	180^{o}
Illumination Zenith Angle	70^{o}
Emission Angle	75^{o}

Table 4.2: Angle constraints to meet the scientific mission requirements of JANUS.

Finally, once the constraints are defined and going back to what was explained at the beginning of the section, this information is passed to *my_tw_finder*, which returns time windows where the specific constraints are met for each ROI.

Then, the function *initializeObservationDataBase* from *oPlanRoi* class is called for each ROI to compute the resolutions, number of images taken and the duration of observation in each time window found, so that these values are saved in class instances.

To perform these calculi, the method *computeObservationData* is called within the function that initializes the observation. This method computes the resolution as shown in Eqs. 4.1 and 4.2, then, from a practical perspective, it is unrealistic to assume that the spacecraft's instrument has an infinite image capture rate, meaning that observations can be considered instantaneous. To address this and allocate sufficient time for each observation, after determining both the resolution and the specific ROI observation times, the program calculates the number of images required to cover the region fully and estimates the total observation duration at each time step in the ROI observation times. Being the process as follows:

- Calculates the surface area that can be captured in a single image based on the current resolution:

$$A_{\text{cov}} = (R \cdot n_{\text{px}})^2 \tag{4.3}$$

  where A_{cov} is the surface area covered in the single image, R the resolution computed as eq. 4.1 and n_{px} the total number of pixels of the instrument's lens, in this case, 1735 pixels.

- Computes the number of images required to cover the region fully:

$$n_{\text{img}} = \frac{A_{\text{cov}}}{A_{\text{ROI}}} \cdot (1 + F) \tag{4.4}$$

  where n_{img} is the number of taken images, A_{ROI} is the total surface area of the ROI and F is a safety factor to ensure a minimum of pictures to be taken.

- Determines the time needed for each observation based on the image rate and the number of images:

$$t = \frac{n_{\text{img}}}{\text{ips}} \tag{4.5}$$

  where t is the time for observation and ips is the image rate, in this case, it will be 0.1 images per

second.

It is worth noting that the calculations do not account for the time variability of resolution. Although observations that are not instantaneous would indeed cause resolution changes over time, necessitating adjustments at each time step, this would also mean that the area covered by each image would change as well, making the observation duration time-dependent, which would increase notably the complexity. However, it is important to balance accuracy and simplicity when dealing with space missions. Assuming that observations happen instantaneously would oversimplify the realities of space missions. Therefore, we assume that the resolution at the beginning of the observation remains constant throughout. Additionally, a safety factor has been introduced to account for any potential underestimation in the number of images required by this simplified model.

Then, after the pre-calculation of these parameters, they are saved together with the intervals start and end for each ROI in a binary file to easily access the data and improve the efficiency of the algorithm.

Finally, as can be seen in Figures 4.3 and 4.4, the number of observations is dramatically reduced due to the applied constraints, making some ROI visible in a few flybys and with not such good resolution than others. In this way, the genetic algorithm will have to decide during which flybys one ROI, should be observed, hoping that it prioritizes those moments in which the resolution is lower, i.e., better.

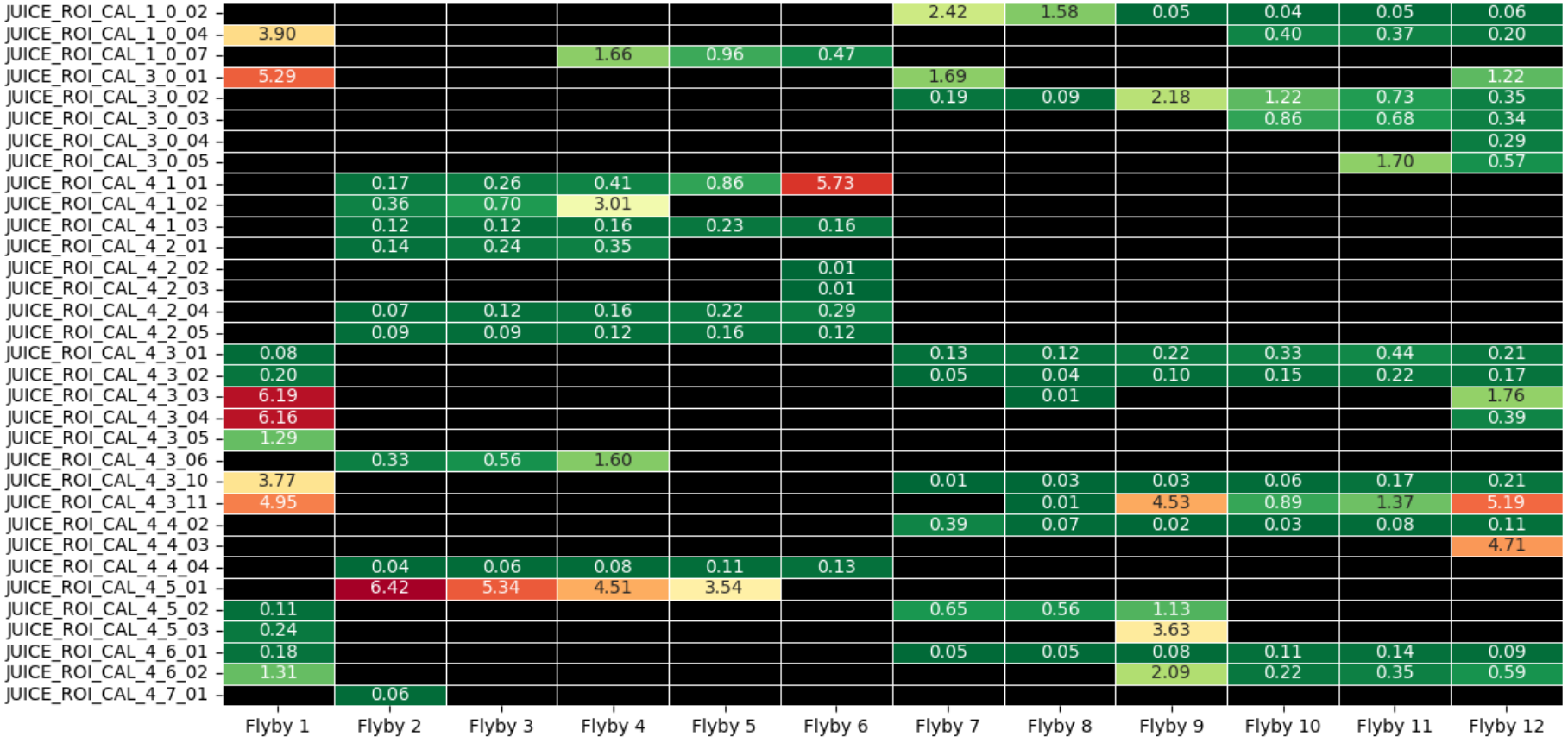


Figure 4.3: This figure shows the **minimum resolution** found along the whole flybys duration for each ROI, computed as described in the previous pages. Part 1.

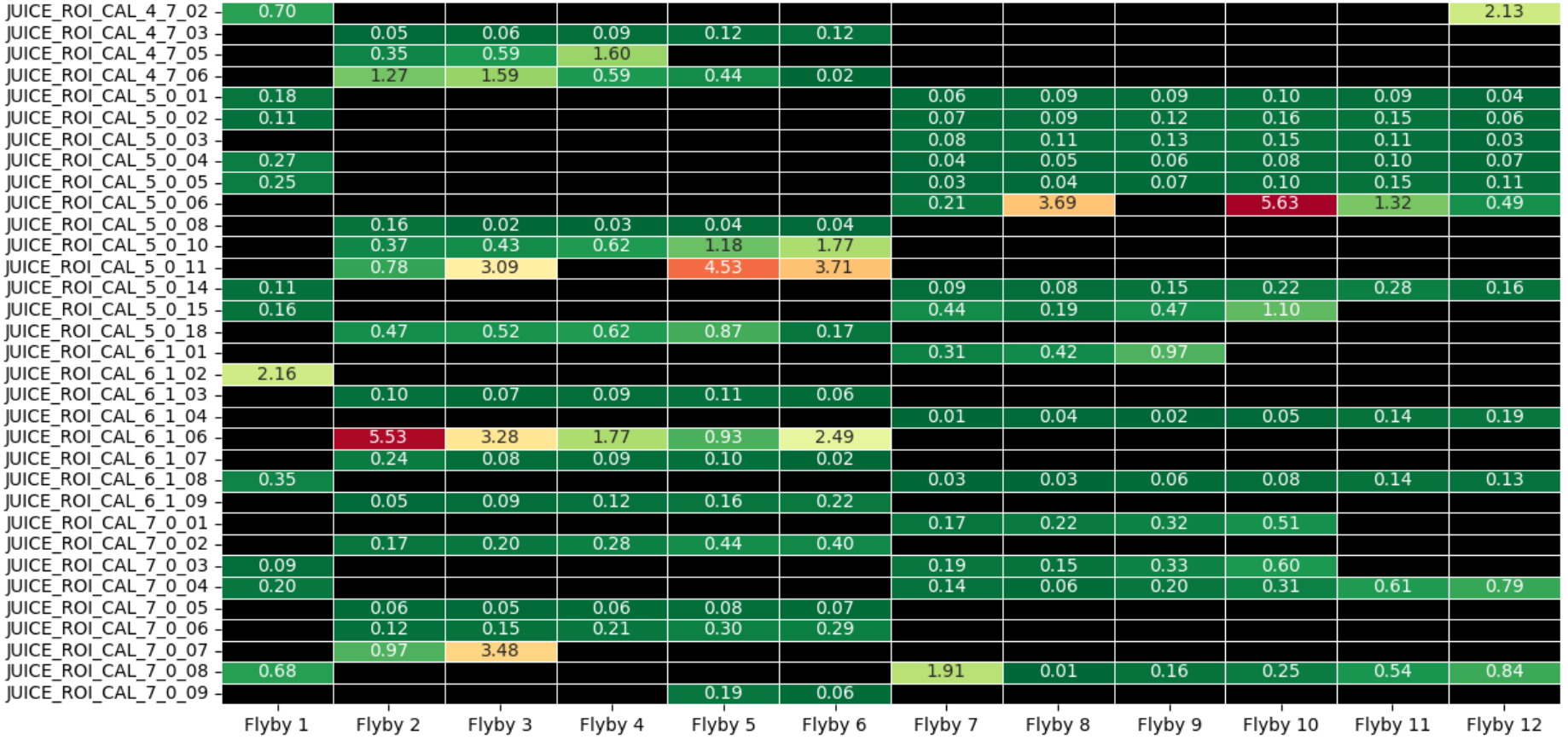


Figure 4.4: This figure shows the **minimum resolution** found along the whole flybys duration for each ROI, computed as described in the previous pages. Part 2.

4.1.3 RIME Opportunity Windows

The same process followed for JANUS is followed to compute the possible scanning opportunities of RIME. In this context, the objective will be to maximize the area covered by the radar, so it will be necessary to compute this coverage. For that, the function *radarcover* from PSOA is used, which computes the distance between the spacecraft and the ground-track point (altitude) and computes the along and across-track as Eqs. 2.1 and 2.2. Then, the area covered as $\rho_{\text{al}} \times \rho_{\text{ac}}$ is returned.

As mentioned, the same process as the observations are followed, hence some constraints must be implemented to meet the scientific specific-requirements of the mission, which can be seen in Table 4.3[1].

In this case, two different environments have to be defined, the *Jovian* side, i.e., the moon's side that faces Jupiter, and the *Anti-Jovian* side, the "far-side" with respect to Jupiter. The reason why is necessary to make this distinction is that the Jovian system emits a high amount of radio signals, thus introducing high levels of noise in the antenna measurements and rendering it incapable of scanning certain regions of the Jovian side [23]. Since Callisto is tidally locked —as Earth's Moon—, the same side of Callisto always faces Jupiter.

Parameter	**Relation**	**Value**
Point in Polygon	Equal	True
Altitude [km]	$<$	1000
3 Points Angle (Jovian)	$<$	135^{o}
3 Points Angle (Anti-Jovian)	$>$	135^{o}

Table 4.3: Angle constraints to meet the scientific mission requirements of RIME.

Then, the function *pointinpoly* determines if the spacecraft is at Jovian or Anti-Jovian side and *compute_3points_angle* computes the angle between JUICE, Callisto and Jupiter, being the constraint different depending on the region where the spacecraft is.

The constraints are introduced and *myTwFinderList* finds the feasible time windows to scan for each region, the *initializeScanDataBase* function is called, which uses the *computeScanData* method to obtain the radar coverage in each time step within the compliant intervals. Finally, this value together with the scanning duration and start for each ROI are saved in a binary file, as done in the JANUS observations.

It is worth noting that this is a simplification of the real problem and objectives of RIME since the quality of the scanned region does not depend only on how large is the coverage but also on how deep it can look, being the latter a variable dependent also on the frequency of the emitted signal and the properties of the moon surface materials.

[1]The actual data of the constraints imposed on this instrument during this part of the mission has been provided by Paolo Capuccio, from European Space Astronomy Centre (ESAC)

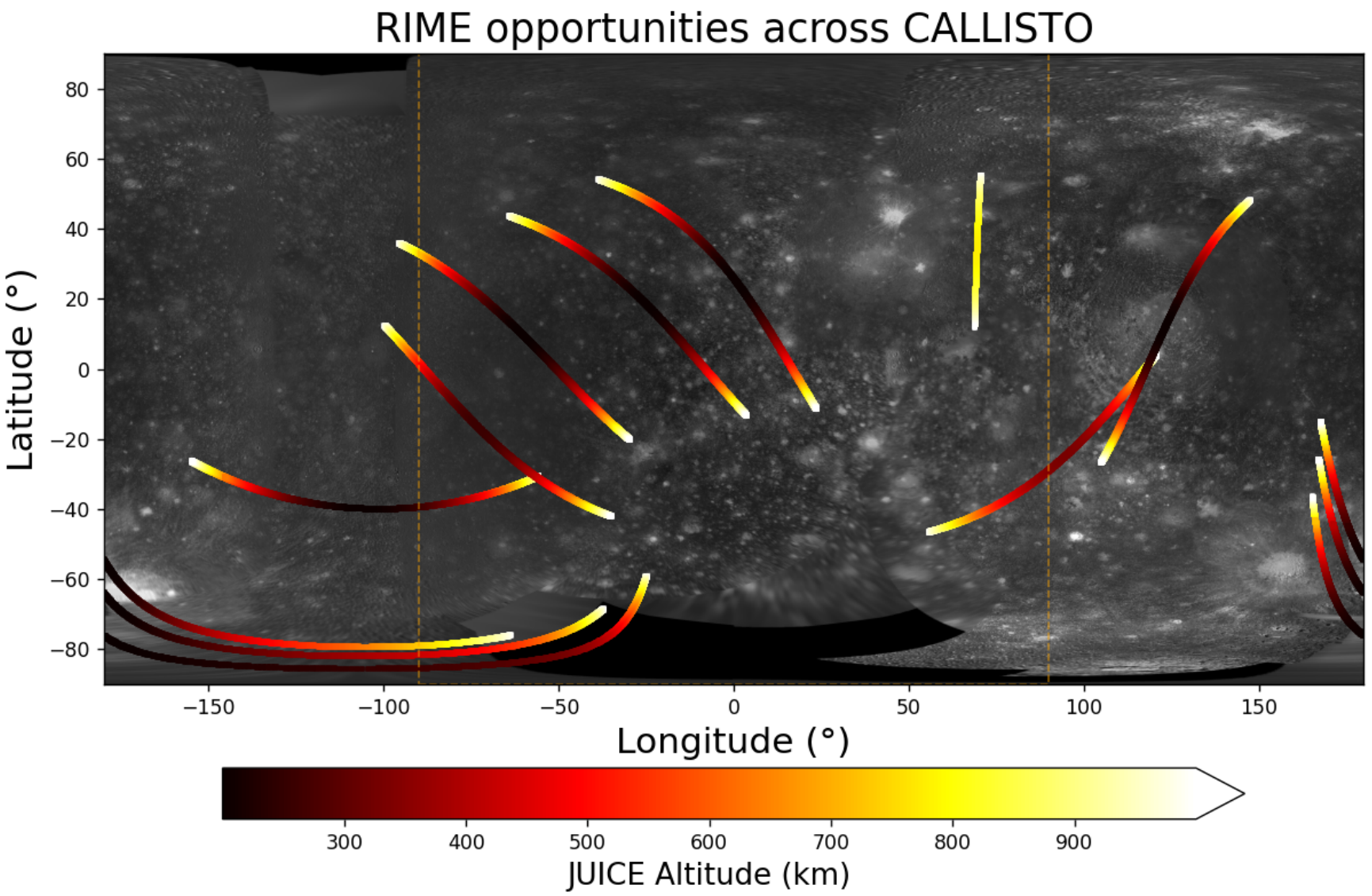


Figure 4.5: This figure shows the different feasible opportunities of scanning that RIME has after applying the specific constraints. The orange dashed line represents the Jovian region, being the zone inside the lines. The spacecraft's altitude is represented due to the direct relation with the scan coverage, the higher the altitude, the greater the area covered.

4.2 Problem Optimization

Along this section, the *Genetic Algorithm* applied to the optimization of the scheduling problem is introduced. So far, the observation opportunity windows have been obtained, now the focus will be on the optimization search, i.e., applying GA to find an optimized observation plan that maximizes the established objectives. Thus, this section will explain how the GA works with the scheduling data.

Firstly, the ROI names are extracted from .txt files with *ROIDataBase* class (see section 4.1). Then, the data from the binary files created previously are loaded comparing the name of the file with the ROI names extracted, ensuring that all the data is loaded. Finally, the loaded data —recall that the start, end and duration of each observation interval, the interval as an array of ephemeris time and the values of the objective function of each instrument for each region were stored— is introduced again in functions *initializeObservationDataBase* and *initializeScanDataBase* from *oPlanRoi* class. In this case, as the data is introduced, it will not be necessary to use the *computeObservationData* method. Then, each variable **roi** is appended to a list of rois, being the ROIs to observe and the regions to scan in different lists. With this data in hand, an instance of *DataManager* class is defined to allow the genetic algorithm to directly access the data without computing it for each individual (see Fig. 4.6).

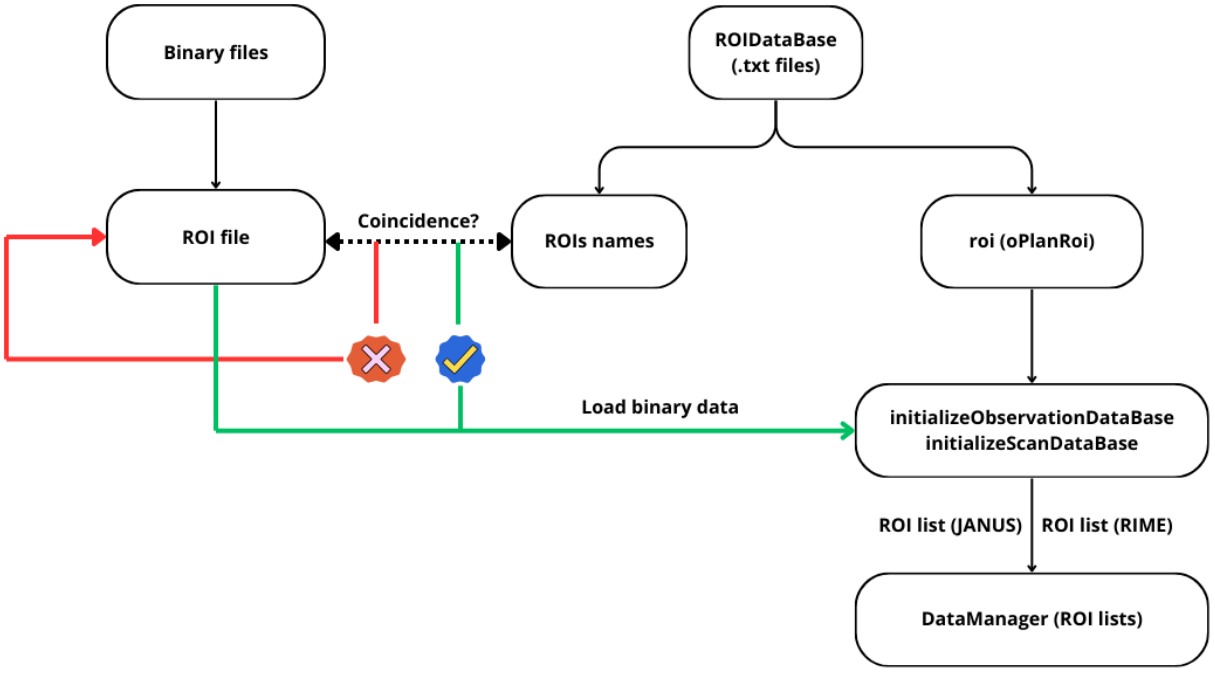


Figure 4.6: Blocks diagram of data flow for easy managing and accessing by the genetic algorithm.

Once the pre-computed data is saved as an instance of *DataManager* class, the *oplan* class is initialized. This class contains all the evolutionary operators, such as *mutFun*, *repFun*, *ranFun* and *fitFun*, in addition to the genotype of the individuals in the population, which will be a set of *oplan* class instances. The instances are the following:

- **.stol**: this instance saves the start of the observation of each ROI.
- **.obsLength**: this instance saves the observation duration of each region.
- **.qroi**: this instance contains the average resolution of each observation interval.

4.2.1 fitFun()

So far, it has been addressed how to compute the JANUS resolution of certain surface points, the area covered by RIME and how to improve the performance of the GAs for multi-objective optimization. Now, the fitness function will be defined as a 2-D function composed of the average camera resolution and the total radar coverage.

For the average resolution two functions are employed; *evalResRoi* and *evalResPlan*. The *evalResPlan* function takes every observation interval as .stol + .obsLength and divides them into 4 time points. Then, calls to *evalResRoi* function for each time. This function computes the resolution at a given time using a linear interpolation between the given time and the already calculated interval since this time will have been modified by the evolutionary operators, so it will not coincide with the start of these pre-computed intervals but instead be inside of them. Then, the average resolution of these 4-time steps is computed for every interval, i.e., for every observable ROI. Finally, the average resolution of the entire set of ROIs is given as the objective function.

Algorithm 10 evalResPlan

```
for i in range length of .stol do
    Set start as .stol[i]
    Set end as .stol[i] + .obsLength[i]
    Set et as numpy.linspace(start, end, 4)
    Define reslist as empty list
    for t in et do
        Append evalResRoi(i, t) to reslist
        Set .qroi[i] as sum(reslist) divided by length of et
    end for
end for
return .qroi
```

Algorithm 11 evalResRoi

```
Input:
    i: Index of observation interval
    et: given time in ephemeris J2000
Set roiL as DataManager.getInstance().getROIList()    ▷ Loads the data from DataManager class
Set res as roiL[i].interpolateObservationData(et)
return res
```

As said before, the instances that define each individual are given by the starts of JANUS observations, the observations duration and the computed resolutions; thus, the scanning intervals are not being taken into account. The reason why it does not appear in the genotype of each individual is that the scan intervals are computed as complementary to the observation intervals, i.e., under the assumption that when JUICE is

taking images, the radar does not work and, on the other hand, when it is scanning, the camera does not make any observation.

To compute these scanning intervals the *computeScanWindows* function is used. This function first defines a SPICE time window in which the different observation intervals from *.stol* and *.obsLength* are introduced. Then, another SPICE time window is created, but, in this case, since there is no any *oplan* class instance that contains the scanning intervals info, it is necessary to access *.ROI_TW* instance from *oPlanRoi* class of each ROI related to the radar, i.e., Jovian and Anti-Jovian sides, loaded with the *DataManager* class. Then, the complementary time window to the feasible scanning intervals and the actual observation intervals given by the individual is computed using the SPICE function *wndifd*. In Fig. 4.7 can be seen an example of the functionality of *spice.wndifd*:

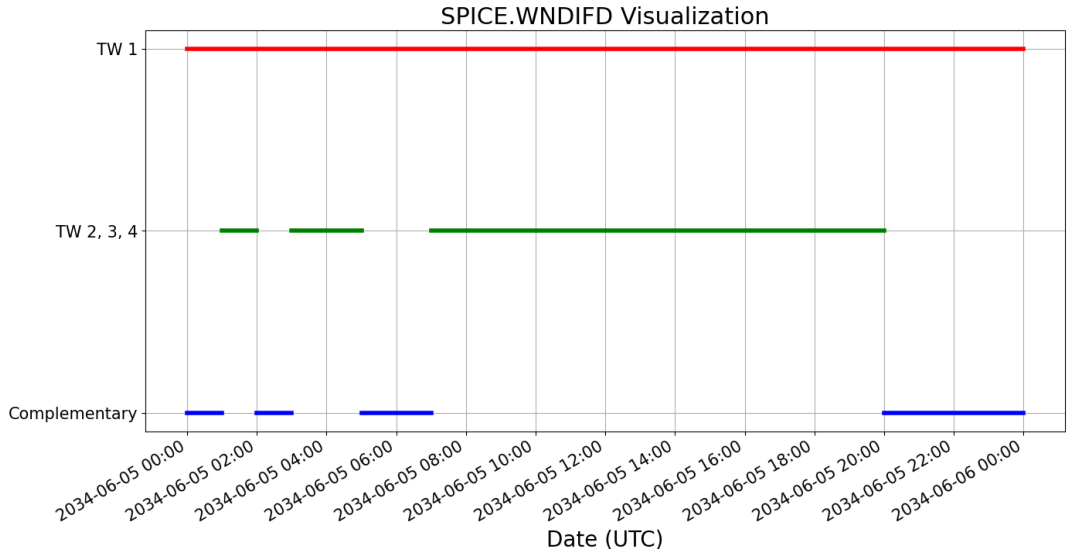


Figure 4.7: This figure explains how SPICE.wndifd works to calculate the complementary time window to two given SPICE time windows. In the context of the problem, TW 1 would be devoted to scanning, while TWs 2, 3 and 4 would be the TW given by the different ROIs devoted to JANUS observations.

Once the scanning time window with the intervals is obtained, the *evalCovScan* function computes the total coverage by following a similar approach as *evalResPlan*, as can be seen in the algorithm 12. Therefore, the fitness function *fitFun* will be

$$\texttt{fitFun} = [\texttt{average of } evalResPlan\ ,\ -evalCovScan]$$

The symbol - is used to maximize the function, since in this case the greater, the better.

So far, it has been shown how to perform the calculations relative to the scheduling problem. Little by little everything has been making sense until the moment in which it is known that each individual is represented by the instances that contain the starts, duration and resolution of each observation and that the fitness function that makes the population to evolve is a list of the average resolution and the total coverage of the scan time window, computed as the complementary of observations time window.

Algorithm 12 evalCovScan

```
Input:
  target: target body to be scanned (Callisto in this thesis)
Set radii as spice.bodvrd(target,'RADII')
Set scan_tw as .computeScanWindow()
Set observer as DataManager.getInstance().getObserver()
Set n as spice.wncard(scan_tw)          ▷ Computes the number of intervals inside the time window
Define totalCov as empty list
for i in range n do
    Set start and end as spice.wnfetd(scan_tw, i) ▷ Gives the start and end points of the interval i of the specific TW
    Set et as numpy.linspace(start, end, 4)
    Define covlist as empty list
    for t in et do
        Append radarcover(radii, groundtrack(observer, t, target), t, target, observer) to covlist
    end for
    Append sum(covlist) divided by length(et) to totalCov
end for
return sum(totalCov)
```

4.2.2 ranFun()

In this sub-section it is shown the initialization of the population and the introduction of newcomers. This process is performed by *ranFun()*, whose basis is to generate individuals randomly.

For that, firstly, the list of ROIs is loaded from *DataManager* class as seen in previous functions. Then, for each roi in the list of rois a function called *uniformRandomInTW()* is used to obtain a new start and duration of observation interval as follows:

- Access to the different observation intervals of the specific ROI.
- Computes a probabilities list to give a high probability of being chosen to the largest interval.
- Selects the interval randomly taking into account the probabilities or weights and gets the start and end of the interval.
- Generates a random number between start and end values and computes the observation length with the *interpolateObservationData* function.
- If the new start —random value between start and end— plus the interpolate length is less or equal to the end of the interval then returns these new values.

Algorithm 13 uniformRandomInTw

```
Input:
  roi: region of interest
Set nint as spice.wncard(roi.ROI_TW)          ▷ Number of intervals in the ROI time window
Define plen as a list of zeros with length nint
Set outOfTW to True
for i in range nint do
    Set intbeg, intend as spice.wnfetd(roi.ROI_TW, i)   ▷ Extract the start and end points of each interval
    Set plen[i] as intend - intbeg
end for
Set total as the sum of plen
Set probabilities as each plen[i] divided by total
Define val as a list of range nint
Set psel as a random selection from val with probability probabilities ▷ Select one interval randomly with probability proportional to its length
Set i0, i1 as spice.wnfetd(roi.ROI_TW, psel)   ▷ Get the start and end of the selected interval
while outOfTW do
    Set rr as a random value uniformly drawn between i0 and i1
    Set obslen as self.getObsLength(roi, rr)
    if rr + obslen is less than or equal to i1 then
        Set outOfTW to False
    end if
end while
return psel, rr, obslen
```

4.2.3 repFun()

The reproduction function in the context of this thesis is just an arithmetic function in which the average value between the two parents is computed due to the complexity of the problem.

- The midpoint observation start from two parents is computed for each *.stol* component.
- It checks whether the new value belongs to some observation interval of the specific ROI related to *.stol*.
- If the new start fits with any interval, the observation length is computed with *interpolateObservationData* function and the values are appended to two new lists *newind* and *newobs*.
- If the new start does not fit with any interval then the *uniformRandomInTw* function is used and the new values are appended to the lists.
- *.stol* and *.obsLength* are replaced by the new lists.

Algorithm 14 repFun

```
Input:
  p1: parent 1
  f1, f2: fitness values for the parents (unused in this algorithm)
Set roiL as DataManager.getInstance().getROIList()
Define newind as an empty list
Define newobs as an empty list
for i in range len(p1.stol1) do
    Set op as (p1.stol1[i] + self.stol1[i]) / 2 ▷ Compute midpoint between parent and current individual for each region
    Set a, _, _ as self.findIntervalInTw(op, roiL[i].ROI_TW)
    if a != -1 then                              ▷ If the midpoint lies within the valid time window
        Append op to newind
        Set obslen as self.getObsLength(roiL[i], op)
        Append obslen to newobs
    else                                         ▷ If the midpoint is outside the valid time window, it is initialized randomly
        Set _, rr, obslen as self.uniformRandomInTw(roiL[i])
        Append rr to newind
        Append obslen to newobs
    end if
end for
Set self.stol1 as newind
Set self.obsLength1 as newobs
```

4.2.4 mutFun()

Finally, the mutation function is presented. As in the same of evolutionary operators, the list of ROIs is loaded to access the pre-computed data. Then, for each ROI in the rois list, the new starts and observations length are computed using the *randomSmallChangeInTw* function, which works as follows:

- Looks if the passed start point lies in any interval of the specific ROI and extracts the start and end points of the respective interval.
- If the given time point is closer to the start, then defines σ as the difference between the time point and the start, if is closer to the end of the interval then just the opposite.
- The new beginning will be set as the given point plus a random value between 0 and the computed σ. The observation length is computed again with *interpolateObservationData*.
- If the new start is greater or equal to the interval start point and the new end is less or equal to the interval end point, then returns these new values.
- If not, computes again the random new value. If it does the process more than 50 times, $\sigma = \sigma/2$, if it does it more than 500 times it raises an exception.

Algorithm 15 randomSmallChangeIntw

```
Input:
  t0: current initial time of the interval
  roi: region of interest
  f: some fitness function (unused in this function)
Set i, intervals, intervalend as self.findIntervalInTw(t0, roi.ROI_TW)
if abs(t0 - intervals) ≥ abs(t0 - intervalend) then
    Set sigma0 as abs(intervalend - t0)
else
    Set sigma0 as abs(intervals - t0)
end if
Set sigma to sigma0
Set ns to 0
while True do
    Set newBegin as t0 + np.random.normal(0, sigma)
    Set obslen as self.getObsLength(roi, newBegin)
    Set newEnd as newBegin + obslen
    if newBegin ≥ intervals and newEnd ≤ intervalend then  ▷ Ensure new interval lies within the current bounds
        break
    end if
    Increment ns by 1
    if ns > 50 then
        Halve sigma0 and set sigma = sigma0
    end if
    if ns > 500 then
        Raise an Exception: 'uhhh cant find mutation'
    end if
end while
return newBegin, obslen
```

With this information in hand, we can now start generating some results using the *mymaga* class since it is a generic class that allows the population evolution, whatever the characteristics of the individuals.

Chapter 5

Results and discussion

The multi-objective optimization presents numerous advantages over the single-objective since it allows to solve different problems with a wide range of difficulties, while taking into account many variables simultaneously, demonstrating its versatility. However, one of the main problems this kind of optimization presents is that it is not easy to visualize. The human brain is able to visualise space in up to 3 dimensions, so the higher the number of dimensions, the more difficult it becomes to interpret the results. Then, although we are in a two-dimensional space, we have seen that a set is formed with the "best" individuals, a.k.a., the first Pareto Front, which allows us to see how good each individual is for each objective, but we do not know what each individual really is, what a change from one individual to another means.

For this reason, during this chapter, we will see how to interpret the different results, generated considering different parameter settings of the genetic algorithm and the observation data to see how they work. Before starting with the results, it is worth noting that, looking closely at the tables in the Appendix of this thesis, we can appreciate that the scan intervals (RIME) are really short. Thus, although the observation intervals (JANUS) are larger, they barely coincide, making hard to find a case where there is a clash of interests in scientific objectives. Therefore, two different cases have been carefully selected in order to show how the algorithm works in the scheduling problem. For these two cases, the configuration that can be seen in Table 5.1 has been implemented.

Population	100
Generations	50
Elites	10
Newcomers	10
Mutants	18
Descendants	62
Can Mutate	15
Can Procreate	15

Table 5.1: Set of parameters for case study 1. This set of parameters uses the same proportion of mutants, descendants, elites and newcomers as the set used for benchmarking tests.

5.1 Case 1

For this first study case, we selected the ROI *JUICE_CAL_4_3_03* and the flyby from *2033 JAN 13 13:32:12* to *2033 JAN 14 13:32:12*, where there is a coincidence in the time windows between JANUS and RIME of six minutes. As mentioned before, the scan intervals are quite short since the constraints limit the opportunity windows to an altitude of less than 1000 km. In consequence, the duration of the feasible scanning windows goes from a couple of minutes to less than twenty minutes as the maximum. Then, when executing the algorithm with the data from this flyby and the unique ROI the following results are obtained.

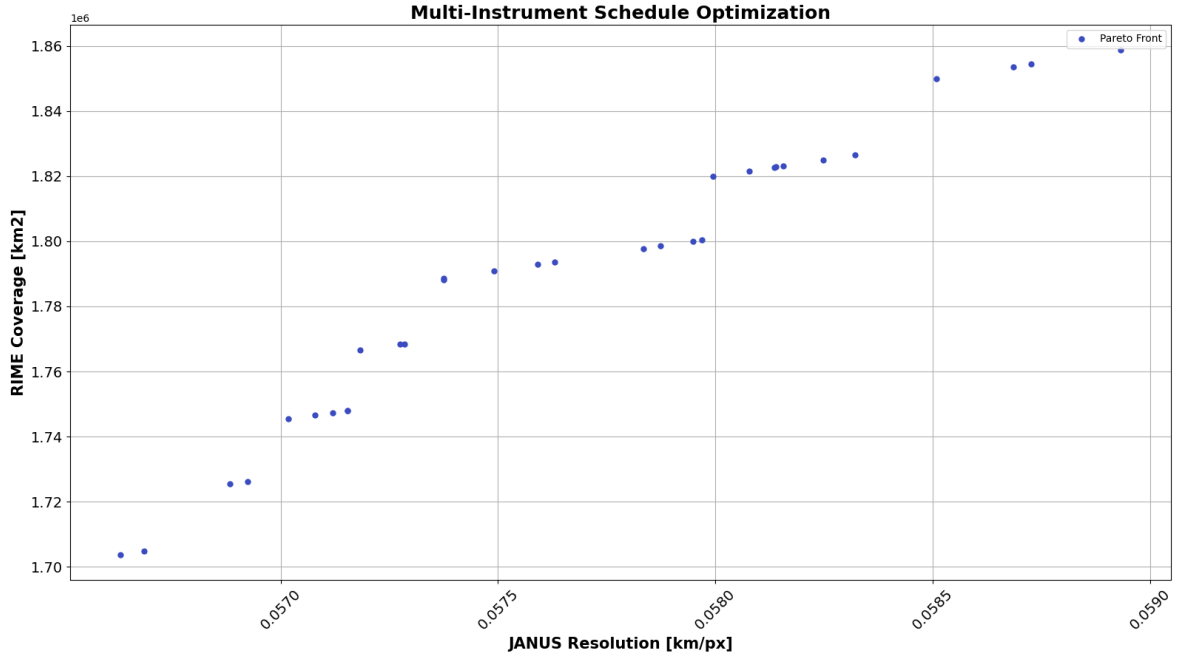


Figure 5.1: First Pareto Front with 32 individuals for Case 1 in which the scheduling problem is optimized for one single ROI and one single flyby.

As we can observe, a large Pareto Front is obtained with 32 individuals, which means that the algorithm is able to find many different options that are equally good in terms of objectives space, giving great flexibility.

However, when looking at this graph we do not understand more than the resolution and coverage value associated with each point that makes up the Pareto Front. Therefore, for a better understanding of what the optimized candidates entail, we look into the first individual and the second —recall that, based on the *Crowding Distance* algorithm, the individuals at the edges of the Pareto are moved forward to the first positions to ensure greater diversity in the population— to see how long is the spacecraft using JANUS and RIME. This will give us an idea of the solution range in which the algorithm stays within.

Individuals	Individual	Individual 2
# Images	21	15
Observation Length [s]	210	150
Scan Length [s]	730.62	774.02
Resolution [km/px]	0.0566	0.0589
Coverage [km2]	1703775.97	1858746.73

Table 5.2: Results obtained for Case 1 for the individuals at the edges of the Pareto Front. One of them is better in resolution objective, while the other one is better in coverage objective.

As seen in Fig. 5.1 and Table 5.2, the algorithm provides numerous possibilities with different distributions of intervals; from one in which the spacecraft prioritizes the observation made by JANUS, thus spending more time taking images, to another option in which the radar RIME is used instead of the camera, reaching a higher value for the covered area. The MOGA successfully elucidates the non-dominated solution space —i.e., the first Pareto front—, which delineates the trade-offs in observation allocation between JANUS and RIME. At the extremes of this front lie the boundary cases, where one prioritizes allocation to JANUS and its targeted ROIs and the other one grants higher coverage to RIME.

5.2 Case 2

A second study case in which more ROIs and flybys have been taken into account. In the previous example, we saw the differences between two individuals who are at the extremes of the Pareto Front. When the problem takes place in a short period, a possible visualization tool could be a *GANNT* chart, where the different time intervals are plotted as boxes along the time axis (x-axis) for each ROI (y-axis). In order to demonstrate the optimization capacities of the MOGA, we increase the problem complexity and dimensionality introducing more ROIs to explore and increasing the range of time to be scheduled, from one flyby to four.

The idea is to plot the ground track over Callisto with different colours depending on which task is doing JUICE at that moment. In this way, it is easy to see the transitions between the different instruments and regions along a larger period.

Thus, the flybys to study are flybys 3, 6, 7 and 8 from Table 4.1. As we did in the previous case, we will look at the obtained results from the first and second individuals, i.e., the ones at the edges of the Pareto Front in Fig. 5.2.

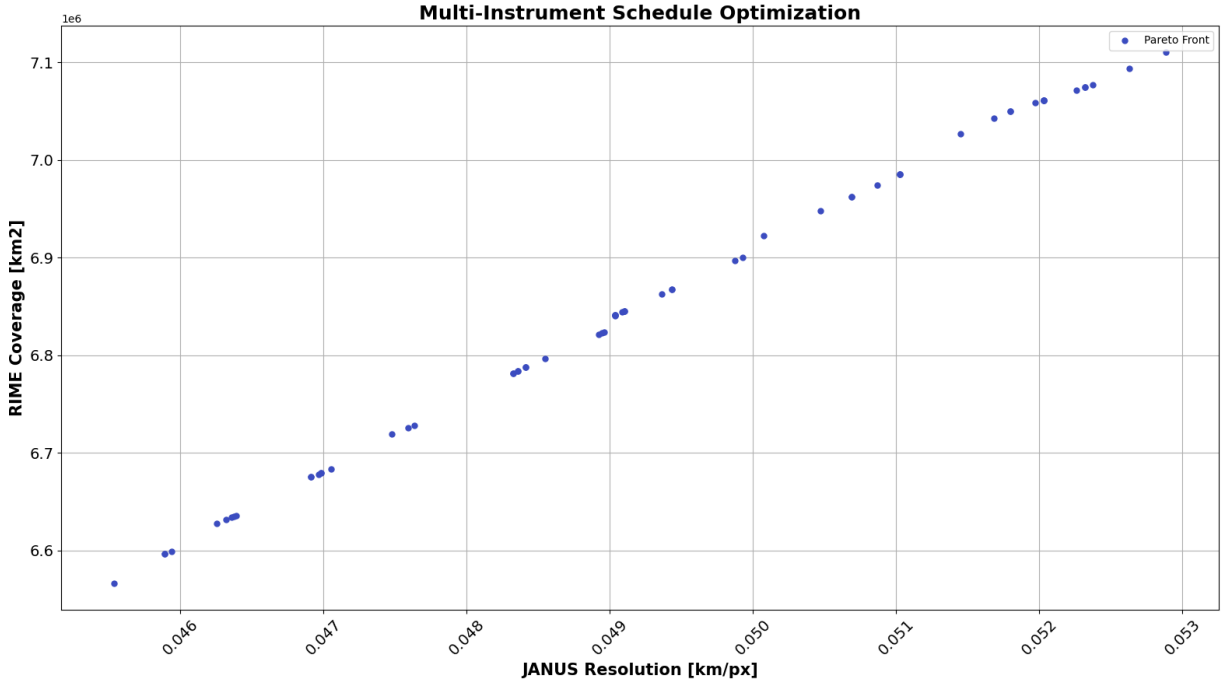


Figure 5.2: First Pareto Front with 63 individuals for Case 2 in which the scheduling problem is optimized for five different ROIs during four Flybys.

As we can observe, in this second case the number of individuals in the Parento Front increases to 63, forming an almost straight line composed in turn of small clusters, suggesting that these small clusters are likely to be combinations in which the observation and scanning intervals vary by a few seconds.

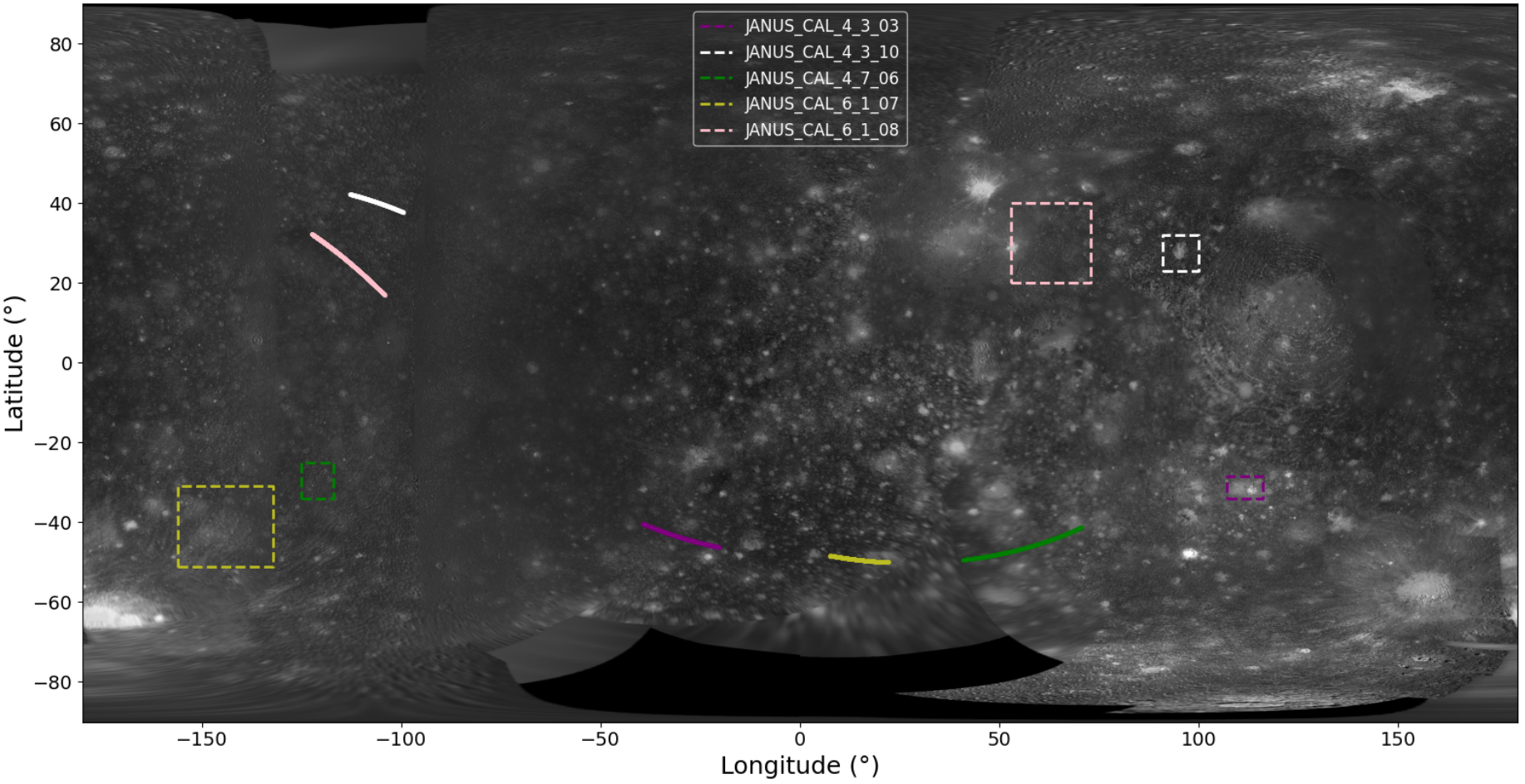


Figure 5.3: This figure shows the JANUS observations from the **first** individual of the Pareto Front, which favours JANUS observations. The colour lines represent the moment in which JANUS is observing at the ROI, represented in dashed lines, with the same colour.

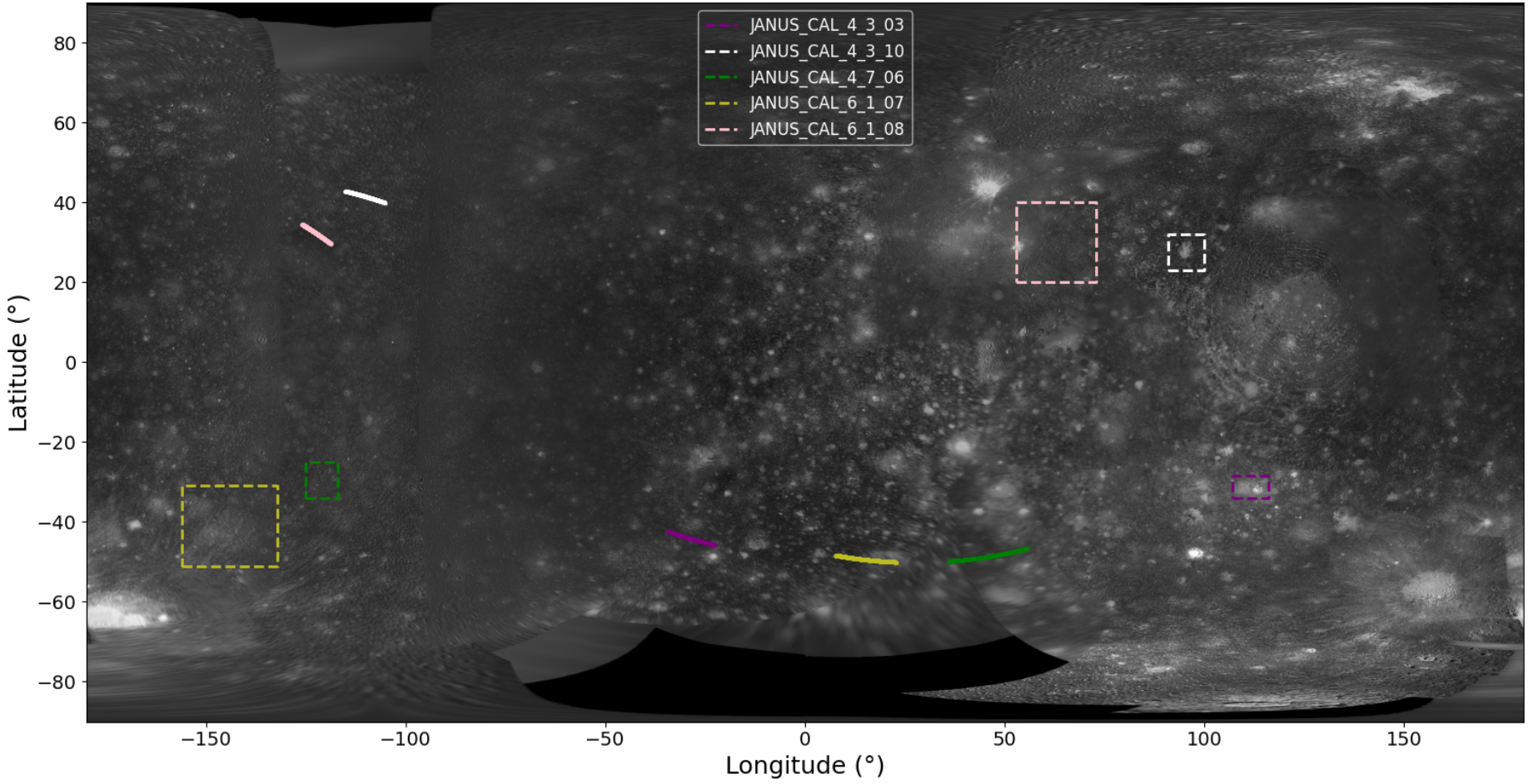


Figure 5.4: This figure shows the JANUS observations from the **second** individual of the Pareto Front, which favours RIME coverage. The colour lines represent the moment in which JANUS is observing at the ROI, represented in dashed lines, with the same colour.

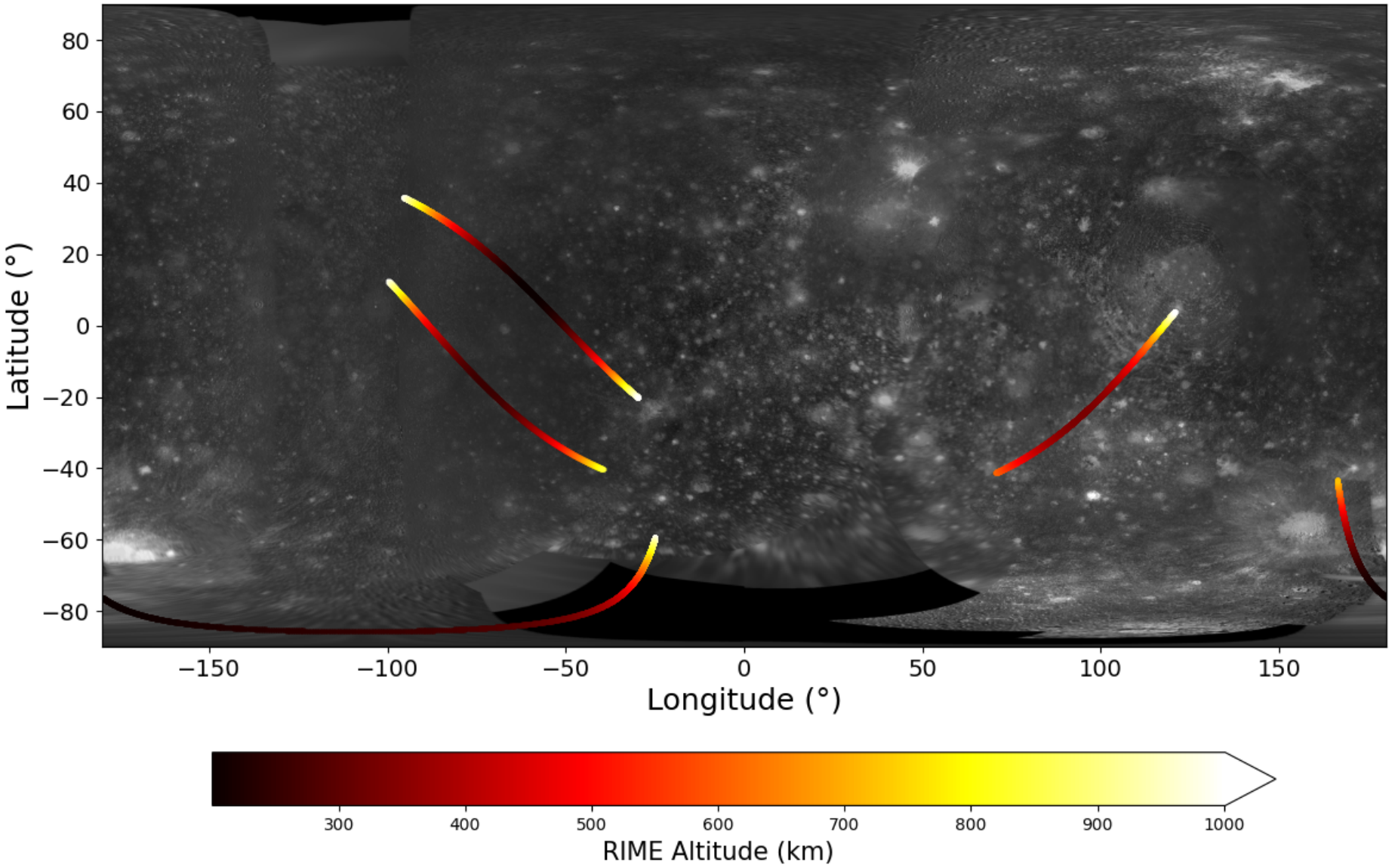


Figure 5.5: This figure shows the RIME scan intervals from the **first** individual of the Pareto Front, which favours JANUS observations. The line is plotted as a colour map depending on the altitude, which is directly correlated to the coverage (see Eqs. 2.1 and 2.2).

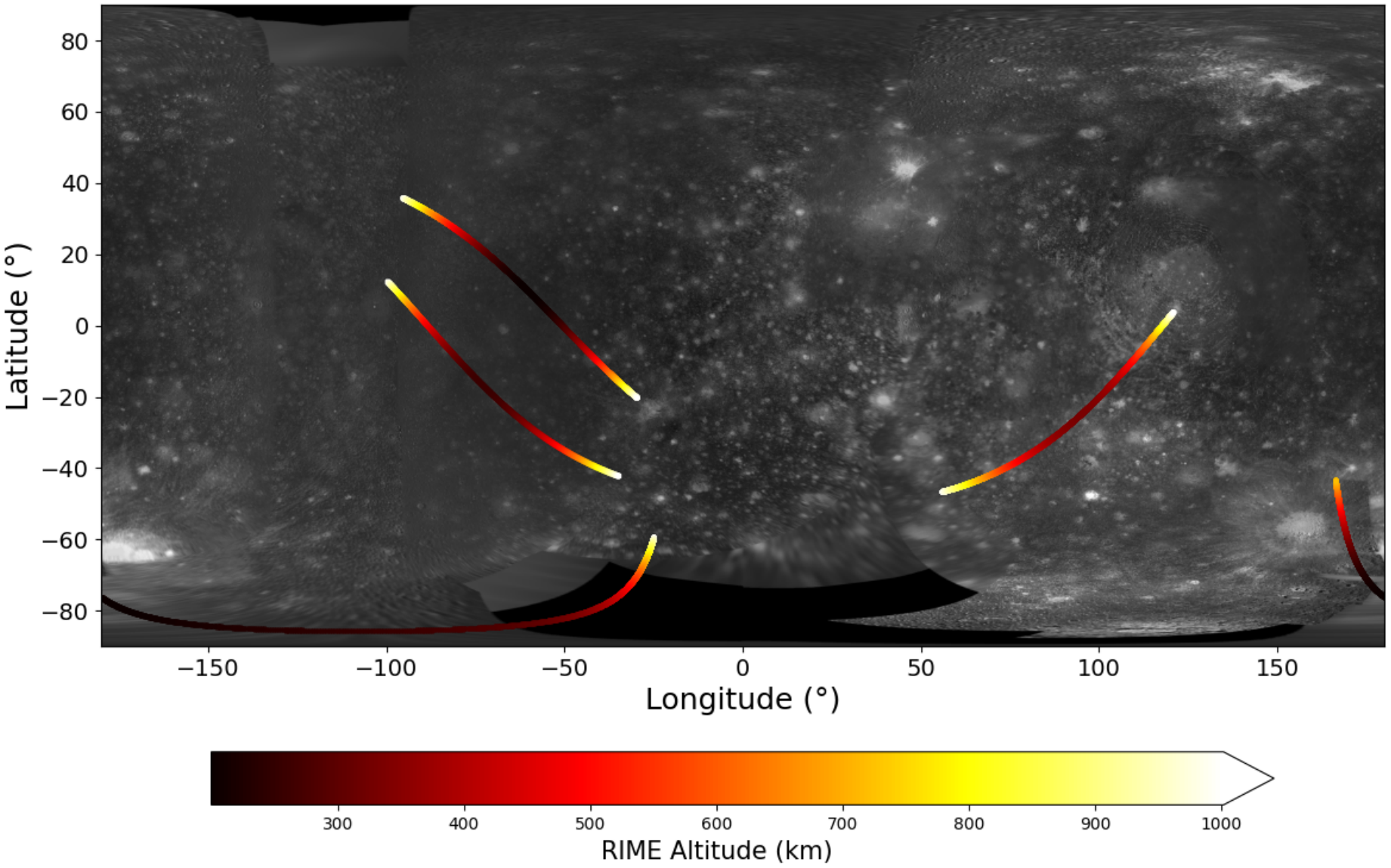


Figure 5.6: This figure shows the RIME scan intervals from the **second** individual of the Pareto Front, which favours RIME coverage. The line is plotted as a colour map depending on the altitude, which is directly correlated to the coverage (see Eqs. 2.1 and 2.2).

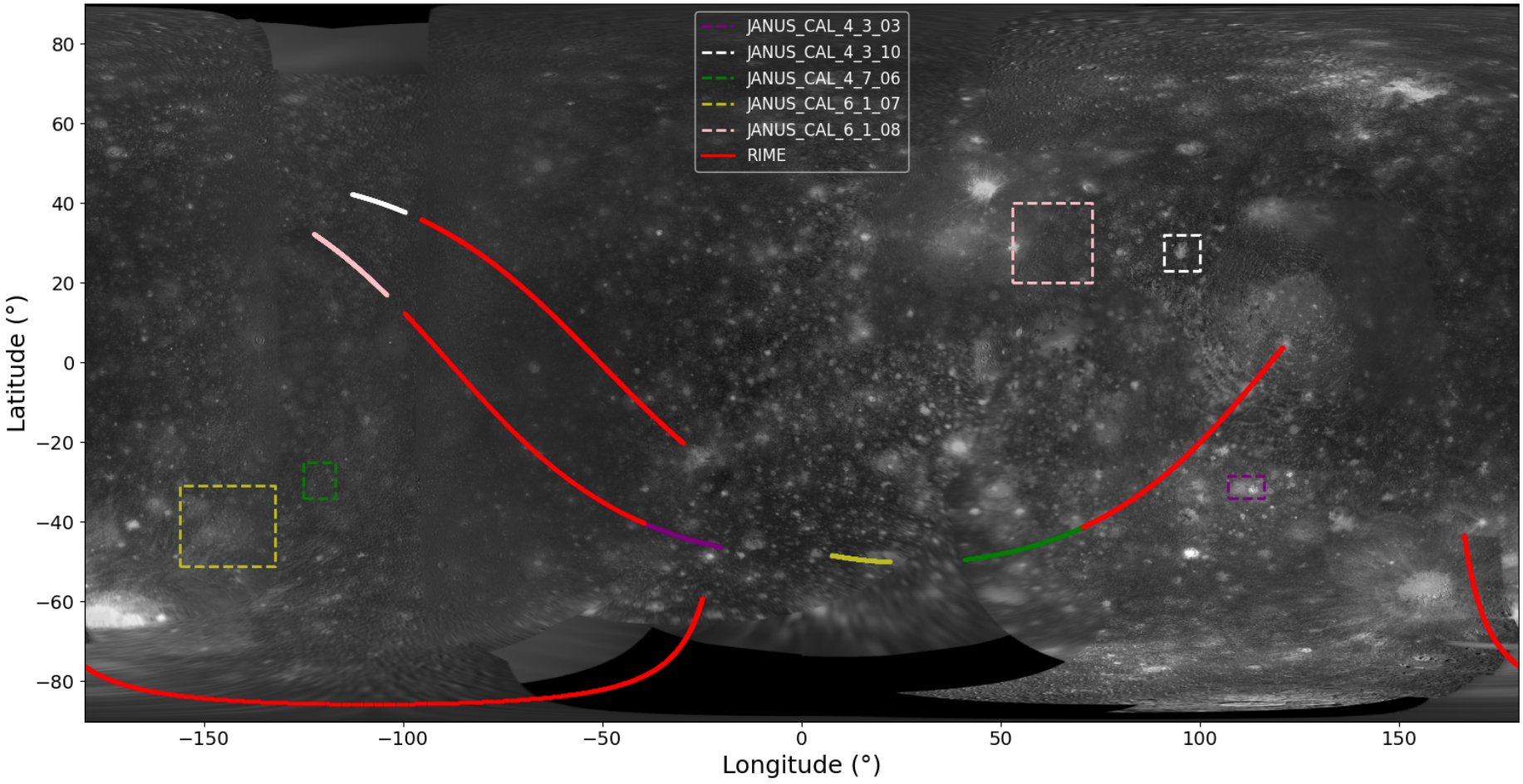


Figure 5.7: This figure shows the JANUS observations and the RIME scan intervals from **first** individual of the Pareto Front together in the same image. While JUICE ground track during JANUS observations is color-coded to show correlation with corresponding ROIs, RIME scanning intervals have been represented with a uniform red line.

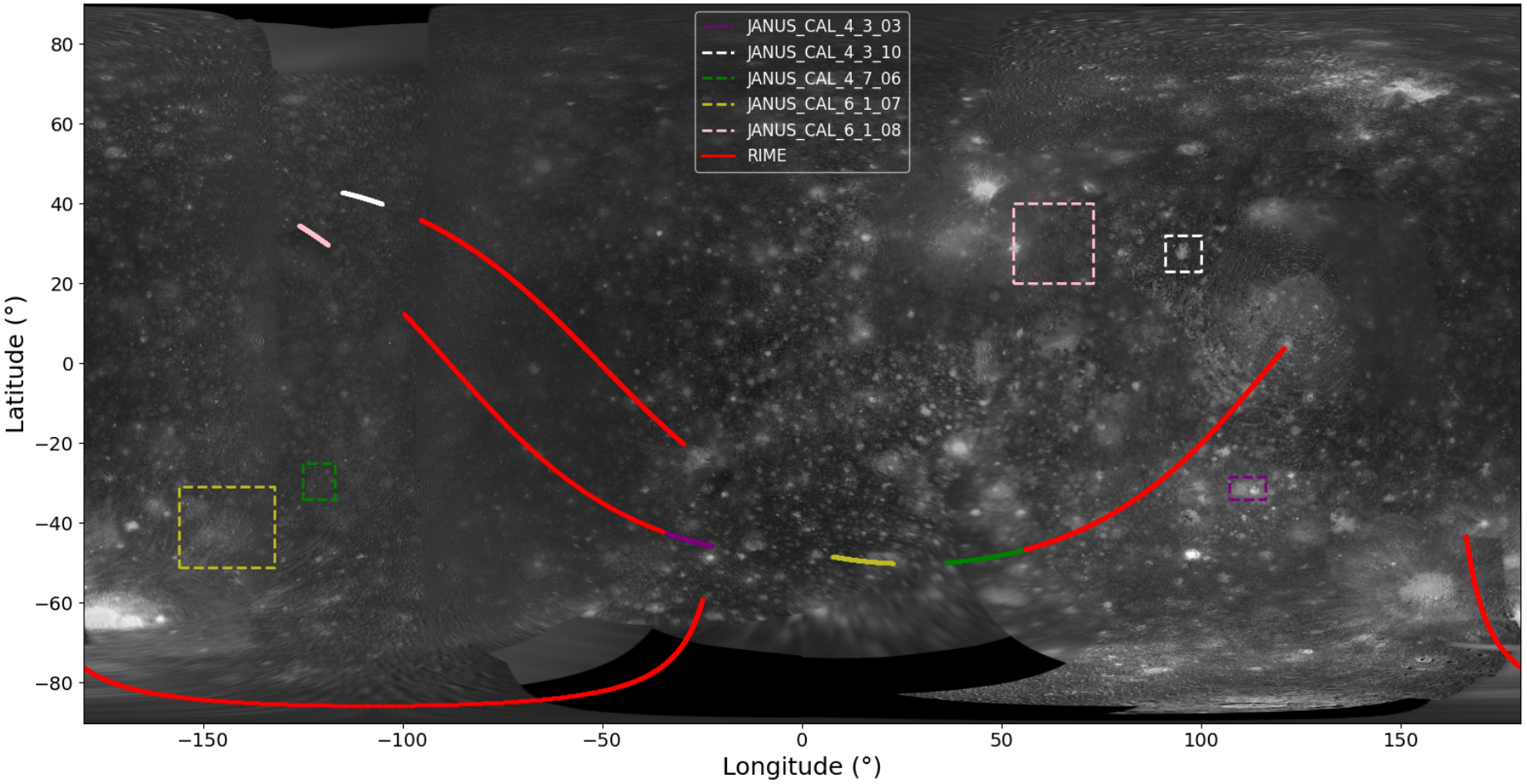


Figure 5.8: This figure shows the JANUS observations and the RIME scan intervals from **second** individual of the Pareto Front together in the same image. While JUICE ground track during JANUS observations is color-coded to show correlation with corresponding ROIs, RIME scanning intervals have been represented with a uniform red line.

As can be seen in Figs. 5.3, 5.4, 5.5, 5.6, 5.7 and 5.8, when looking at the observations of individual 1 we can appreciate larger observations for JANUS, so it is expected that this individual spends more time taking images and therefore achieve a better resolution. Still, it spends less time scanning with RIME, so the total covered area by the radar is smaller. The opposite happens to individual 2, when looking at its figures we see that the JANUS observations are shorter, so it spends less time observing the different ROIs, thus it achieves worse resolution, whereas the global area covered by RIME is greater. These facts are corroborated by extracting the observation data for each of the individuals, which can be seen in the following table:

Individuals	Individual	Individual 2
# Images	177	134
Observation Length [s]	1745.01	1312.41
Scan Length [s]	2844.79	3029.76
Resolution [km/px]	0.0455	0.0529
Coverage [km2]	6566661.38	7110556.32

Table 5.3: Results obtained for Case 2 for the individuals at the edges of the Pareto Front. One of them is better in resolution objective, while the other one is better in coverage objective, as seen in the previous figures. The shown values are the total values for all the ROIs.

In the first case study, the results reveal a diverse range of optimal solutions found by the MOGA, as evidenced by the 32 distinct individuals forming the first Pareto front. The spread along the Pareto front (Fig. 5.1) reflects the trade-off between prioritizing JANUS observations and RIME scanning. The first individual is optimized for image resolution, capturing 21 images with an average resolution of 0.0566 km/px, while the second individual sacrifices some resolution (0.0589 km/px) to maximize radar coverage with RIME, achieving a higher covered area (1.86 million km^2 compared to 1.70 million km^2). The ability of the algorithm to provide such a variety of solutions demonstrates its robustness in balancing scientific objectives that are sometimes in conflict. Specifically, while one solution favors detailed imaging, another optimizes for extensive radar scanning, offering mission planners flexibility depending on the mission's evolving priorities.

In the second case study, which introduces more complexity by considering multiple ROIs over four flybys, the first Pareto front grows to 63 individuals (Fig. 5.2). The linearity and clustering observed in this front indicate that subtle variations in the observation and scan timing across multiple flybys result in small but meaningful changes in mission performance. In Figs. 5.3 and 5.5, the first individual is biased toward longer JANUS observation times, which results in higher average global image resolution (0.0455 km/px) but slightly reduced global radar coverage (6.57 million km^2). Conversely, the second individual (Figs. 5.4 and 5.6) favors RIME radar scanning, covering a larger area of 7.11 million km^2 but at the cost of slightly worse average image resolution (0.0529 km/px).

The ability of the MOGA to explore this solution space is particularly evident in the visualizations (Figs. 5.7 and 5.7), where transitions between different instruments across the flybys are color-coded. This visual representation underscores how the algorithm optimizes the timing and allocation of observational resources,

ensuring that both JANUS and RIME are effectively utilized without overlap or wasted time. The careful distribution of observation windows ensures that the overall scientific return is maximized, either by focusing on high-resolution imaging or extensive radar coverage, as seen in the extracted data in Table 5.3. This balance between objectives highlights the importance of maintaining diversity in the Pareto front, as it allows mission planners to select a solution that best fits the evolving mission goals.

Chapter 6

Budget summary

Table 6.1: Master's Thesis budget dissertation.

Category	**Description**	**Cost (€)**
Staff costs		
Junior Engineer	350 hours @ 22.5 €/h	7,875.0
Software and Licensing		
Matlab License	Annual Cost - Provided by Polytechnic University of Catalonia (UPC)	0.0
Total Estimated Budget:		**7,875.0**

The total budget of the Master's Thesis is presented. In Table 6.1, a detailed breakdown of the budget is provided. The table is divided into three columns named "Category", "Description" and "Cost" with three sections: Staff Costs, Software and Licensing and Total Estimated Budget. The electricity cost was included in the Junior Engineer costs, assuming that the basis cost is 17.5 €/h and 5.0 €/h are added due to equipment, travel costs and electricity consumption.

Chapter 7

Analysis and assessment of environmental and social implications

The exploration of space, particularly planetary missions like JUICE, requires the development of complex engineering solutions that span across a variety of scientific disciplines. While these missions primarily focus on advancing our understanding of the universe, they also have significant environmental and social implications. In this chapter, a brief analysis of how this study, which focuses on optimizing the schedule of JUICE's JANUS camera and RIME radar, considers environmental and social dimensions will be discussed. Additionally, the potential positive impacts in these areas will be assessed.

Environmental Considerations

Space missions have an inherent environmental impact, particularly through the launch process and the manufacturing of spacecraft. However, the JUICE mission and the methods explored in this thesis offer several considerations for mitigating such impacts.

- **Optimizing Resource Usage**

The MOGA employed in this thesis helps optimize the scientific yield of the mission by carefully planning observation schedules and maximizing the use of available resources. This approach leads to more efficient use of power, instruments, and data acquisition capabilities, thereby reducing unnecessary resource consumption. In particular, by maximizing the quality of observations through careful scheduling of the JANUS camera and RIME radar, the mission can achieve more with fewer observational windows, reducing energy consumption over time. This is particularly important for long-duration missions where energy, provided by solar panels, is limited.

- **Minimizing Data Transmission Overheads**

One of the key environmental impacts of space missions is the energy required for data transmission between the spacecraft and Earth. By optimizing the schedule to focus on periods where the highest quality data

can be collected, the volume of data that needs to be transmitted can be managed more efficiently. This reduces the need for continuous data transmission, lowering the energy footprint of both the spacecraft and the ground-based infrastructure needed to receive and process the data.

- **Sustainability in Space Debris Management**

While this thesis does not directly address space debris, the optimization of mission activities can contribute to sustainability efforts in space exploration. By improving the operational efficiency of the JUICE mission, there is less risk of malfunctions or unplanned collisions that could lead to space debris. In the broader context, ensuring that missions like JUICE are successful in their objectives can reduce the need for redundant missions, which in turn helps limit the amount of material left in orbit.

Social Implications

Space missions also have significant social dimensions, especially regarding the advancement of knowledge, education, and the potential for inspiring future generations. This thesis indirectly contributes to these dimensions through its focus on improving the efficiency and success of space exploration.

- **Contributing to Scientific Knowledge**

The optimization strategies presented in this thesis aim to maximize the scientific return of the JUICE mission, particularly in its study of Callisto. By improving the efficiency of observations and data collection, the mission can generate higher-quality data that will benefit researchers in planetary science, astronomy, and space exploration. The increased understanding of Jupiter's moons has the potential to shape future missions and contribute to our knowledge of the solar system, which has broader educational and scientific benefits for society.

- **Promoting Technological Innovation**

The development of MOGAs for complex scheduling problems in space missions not only improves the immediate success of the JUICE mission but also promotes the advancement of optimization techniques that can be applied to other sectors. The technological innovation involved in solving these scheduling problems can have far-reaching effects on industries such as logistics, energy management, and even healthcare, where efficient resource allocation is critical. The social impact of such technological advancements is substantial, as they can improve quality of life by optimizing resource use in various fields.

- **Inspiring Future Generations**

Space missions like JUICE are often seen as some of humanity's greatest endeavors, capturing the imagination of people worldwide. The success of such missions, made possible through advanced optimization techniques like those explored in this thesis, plays a role in inspiring future generations of scientists, engineers, and explorers. The societal impact of encouraging young people to pursue careers in STEM (Science, Technology, Engineering, and Mathematics) fields cannot be understated, as it fosters innovation and contributes to long-term societal progress.

Chapter 8

Conclusions and future work

This thesis explored the application of GAs to optimize the scheduling of remote sensing observations for the JUICE mission's JANUS camera and RIME radar during flybys of Callisto. The objective was to find optimal observation and scanning schedules that maximize the resolution of images captured by JANUS and the coverage area scanned by RIME while satisfying the mission's geometric and operational constraints.

The initial sections of this work presented an overview of the multi-objective optimization framework, including the genetic algorithm's structure and its operators: mutation, crossover, and selection. The use of Pareto fronts allowed for the simultaneous optimization of competing objectives —resolution and coverage— giving the algorithm the flexibility to balance trade-offs between these two goals.

The methodology detailed the integration of different classes responsible for handling the mission's technical constraints, the pre-calculation of opportunity windows, and the implementation of MOGAs. The introduction of the Crowding Distance method to preserve diversity within the population helped avoid premature convergence, leading to better exploration of the solution space. Additionally, the observation schedule for JANUS and the scanning intervals for RIME were simulated to test the effectiveness of the scheduling algorithm under real mission conditions, confirming its reliability in dealing with the constraints imposed by planetary flybys.

Two case studies were conducted to demonstrate the algorithm's performance. The first case focused on a single ROI and flyby and showed how the algorithm successfully balanced observation and scanning tasks within the limited six-minute window. Results revealed multiple valid solutions, each representing a unique trade-off between JANUS image resolution and RIME radar coverage, as highlighted by the individuals at the extremes of the Pareto front. On the other hand, the second case extended the scope by incorporating multiple ROIs and flybys. Here, the algorithm had to optimize schedules across several regions over multiple flybys, demonstrating the algorithm's ability to handle more complex scheduling scenarios. The visual representation of the ground tracks provided insights into how the genetic algorithm adjusted observation and scanning priorities for different regions, with one individual prioritizing JANUS observations and another maximizing RIME coverage.

The simulations of JANUS observations and RIME scanning confirmed the algorithm's ability to efficiently balance these conflicting objectives, further reinforcing its practical utility. The success of the algorithm implementation lies in its capability to offer mission planners a wide array of potential schedules, with each solution effectively balancing the mission's competing scientific objectives. This achievement demonstrates the robustness of the algorithm in addressing scheduling problems under real-world conditions, where constraints such as short observation windows and limited operational timeframes are critical factors.

The results from both cases confirm the effectiveness of GAs in solving the scheduling problem for remote sensing missions like JUICE. The algorithm provided a wide array of potential schedules, each balancing the competing objectives in different ways, offering mission planners valuable flexibility in their decision-making. The success of this implementation underscores the strength of genetic algorithms as a tool for handling complex multi-objective optimization in space exploration.

While the solutions achieved demonstrate the feasibility of using GAs for scheduling optimization in space missions, future work could further refine the model by incorporating additional constraints, such as energy consumption and data transmission limits. Moreover, the exploration of other optimization techniques, like hybrid methods combining GAs with machine learning approaches, could enhance the efficiency of the solution process, especially for highly complex missions with a larger number of variables.

In conclusion, this thesis has successfully demonstrated the application of Genetic Algorithms in optimizing observation schedules for the JUICE mission, highlighting the robustness, flexibility, and potential of multi-objective optimization techniques for space exploration missions. The successful simulations of JANUS and RIME operations further validate the algorithm's capability to meet mission objectives under challenging conditions, marking a significant achievement in the optimization of space mission schedules.

Bibliography

1. GAO, Yang; CHIEN, Steve. Review on space robotics: Toward top-level science through space exploration. *Science Robotics*. 2017, vol. 2, no. 7, eaan5074.
2. MCNUTT JR, RL. Solar system exploration: A vision for the next hundred years. In: *55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*. 2004, IAA–3.
3. SANGUINO, Tomás de J Mateo. 50 years of rovers for planetary exploration: A retrospective review for future directions. *Robotics and Autonomous Systems*. 2017, vol. 94, pp. 172–185.
4. LI, Xiuhong; SUN, Chongxiang; FAN, Huilong; YANG, Jiale. Remote-Sensing Satellite Mission Scheduling Optimisation Method under Dynamic Mission Priorities. *Mathematics*. 2024, vol. 12, no. 11, p. 1704.
5. COSTA, Antonio; CAPPADONNA, Fulvio Antonio; FICHERA, Sergio. A novel genetic algorithm for the hybrid flow shop scheduling with parallel batching and eligibility constraints. *The International Journal of Advanced Manufacturing Technology*. 2014, vol. 75, pp. 833–847.
6. MINELLA, Gerardo; RUIZ, Rubén; CIAVOTTA, Michele. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*. 2008, vol. 20, no. 3, pp. 451–471.
7. KONAK, Abdullah; COIT, David W; SMITH, Alice E. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability engineering & system safety*. 2006, vol. 91, no. 9, pp. 992–1007.
8. GRASSET, Olivier; DOUGHERTY, MK; COUSTENIS, A; BUNCE, EJ; ERD, C; TITOV, D; BLANC, M; COATES, A; DROSSART, P; FLETCHER, LN, et al. JUpiter ICy moons Explorer (JUICE): An ESA mission to orbit Ganymede and to characterise the Jupiter system. *Planetary and Space Science*. 2013, vol. 78, pp. 1–21.
9. ACTON, Charles. An overview of SPICE. *Jet Propulsion Laboratory: Oak Grove, KY, USA*. 1998.
10. GALUZIN, Vladimir; KUTOMANOV, A.; MATYUSHIN, M.; SKOBELEV, Petr. A Review of Modern Methods for Planning and Scheduling of the Operations in Advanced Space Systems. *Mekhatronika, Avtomatizatsiya, Upravlenie*. 2020, vol. 21, pp. 639–650. Available from DOI: 10.17587/mau.21.639-650.
11. PLAS, Peter. MAPPS: a Science Planning tool supporting the ESA Solar System Missions. In: 2016. Available from DOI: 10.2514/6.2016-2512.
12. CHOO, Teck H; MURCHIE, Scott L; BEDINI, Peter D; STEELE, R Josh; SKURA, Joseph P; NGUYEN, Lillian; NAIR, Hari; LUCKS, Michael; BERMAN, Alice F; MCGOVERN, James A, et al. SciBox, an

end-to-end automated science planning and commanding system. *Acta Astronautica*. 2014, vol. 93, pp. 490–496.

13. AI-CHANG, Mitchell; BRESINA, John; CHAREST, Len; CHASE, Adam; HSU, JC-J; JONSSON, Ari; KANEFSKY, Bob; MORRIS, Paul; RAJAN, Kanna; YGLESIAS, Jeffrey, et al. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*. 2004, vol. 19, no. 1, pp. 8–12.
14. MALDAGUE, Pierre F; WISSLER, Steven S; LENDA, Matthew D; FINNERTY, Daniel F. APGEN scheduling: 15 years of Experience in Planning Automation. In: *SpaceOps 2014 Conference*. 2014, p. 1809.
15. CHIEN, Steve; RABIDEAU, Gregg; KNIGHT, Russell; SHERWOOD, Robert; ENGELHARDT, Barbara; MUTZ, Darren; ESTLIN, Tara; SMITH, Benjamin; FISHER, Forest; BARRETT, T, et al. ASPEN-Automating space mission operations using automated planning and scheduling. In: *SpaceOps 2000*. AIAA Press, 2000.
16. JET PROPULSION LABORATORY ARTIFICIAL INTELLIGENCE GROUP. *CASPER (Continuous Activity Scheduling Planning Execution and Replanning)* [https://ai.jpl.nasa.gov/public/projects/casper/]. [N.d.]. Accessed: 2024-09-12.
17. JET PROPULSION LABORATORY ARTIFICIAL INTELLIGENCE GROUP. *Eagle Eye: Continuous Planning for Earth Observation Satellites* [https://ai.jpl.nasa.gov/public/projects/eagle-eye/]. [N.d.]. Accessed: 2024-09-12.
18. KNIGHT, Russell; HU, Steven. Compressed large-scale activity scheduling and planning (clasp) applied to desdyni. In: *Proceedings of the Sixth International Workshop in Planning and Scheduling for Space, Pasadena, CA*. 2009.
19. (ESA), European Space Agency. *JUICE Mission Patch* [https://scifleet.esa.int/downloads/juice/juice.png]. 2024. Available also from: https://scifleet.esa.int/downloads/juice/juice.png. Accessed: 2024-09-12.
20. TEAM, JUICE Science Working et al. JUpiter ICy Moons Explorer-Exploring the Emergence of Habitable Worlds around Gas Giants. *Definition Study Report ESA/SRE*. 2014, vol. 1.
21. EUROPEAN SPACE AGENCY. *Juice's Journey and Jupiter System Tour*. 2022. Available also from: https://www.esa.int/ESA_Multimedia/Videos/2022/03/Juice_s_journey_and_Jupiter_system_tour. Accessed: 2023-09-18.
22. BRUZZONE, L.; PLAUT, J.J.; ALBERTI, G.; BLANKENSHIP, D.D.; BOVOLO, F.; CAMPBELL, B.A.; FERRO, A.; GIM, Y.; KOFMAN, W.; KOMATSU, G.; MCKINNON, W.; MITRI, G.; OROSEI, R.; PATTERSON, G.W.; PLETTEMEIER, D.; SEU, R. RIME: Radar for Icy Moon Exploration. In: *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*. 2013, pp. 3907–3910. Available from DOI: 10.1109/IGARSS.2013.6723686.
23. BRUZZONE, Lorenzo; ALBERTI, Giovanni; CATALLO, Claudio; FERRO, Adamo; KOFMAN, Wlodek; OROSEI, Roberto. Subsurface Radar Sounding of the Jovian Moon Ganymede. *Proceedings of the IEEE*. 2011, vol. 99, no. 5, pp. 837–857. Available from DOI: 10.1109/JPROC.2011.2108990.

24. GERMAN AEROSPACE CENTER (DLR). *JUICE: JANUS Camera to Investigate Jupiter's Icy Moons*. 2023. Available also from: https://www.dlr.de/en/research-and-transfer/projects-and-missions/juice/janus. Accessed: 2024-09-18.
25. CORTE, Vincenzo Della; NOCI, Giovanni; TURELLA, Andrea; PAOLINETTI, Riccardo; ZUSI, Michele; MICHAELIS, Harald; SOMAN, Matthew; DEBEI, Stefano; CASTRO, Jose Maria; HERRANZ, Miguel; AMOROSO, Marilena; CASTRONUOVO, Marco; HOLLAND, Andrew; LARA, Luisa Maria; JAUMANN, Ralf; PALUMBO, Pasquale. Scientific objectives of JANUS Instrument onboard JUICE mission and key technical solutions for its Optical Head. In: *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. 2019, pp. 324–329. Available from DOI: 10.1109/MetroAeroSpace.2019.8869584.
26. INSTITUTO DE ASTROFÍSICA DE ANDALUCÍA. *La misión JUICE despega con destino Júpiter*. 2023. Available also from: https://www.iaa.es/noticias/mision-juice-despega-con-destino-jupiter. Accessed: 2024-09-18.
27. FORREST, Stephanie. Genetic algorithms. *ACM computing surveys (CSUR)*. 1996, vol. 28, no. 1, pp. 77–80.
28. GOLBERG, David E. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*. 1989, vol. 1989, no. 102, p. 36.
29. EIBEN, A.; SMITH, Jim. *Introduction To Evolutionary Computing*. Vol. 45. 2003. ISBN 978-3-642-07285-7. Available from DOI: 10.1007/978-3-662-05094-1.
30. LUDWIG, Simone. Anthony Brabazon, Michael O'Neill, Sean McGarraghy: Natural computing algorithms. *Genetic Programming and Evolvable Machines*. 2016, vol. 17. Available from DOI: 10.1007/s10710-016-9266-8.
31. EIBEN, A.; SCHIPPERS, C. On Evolutionary Exploration and Exploitation. *Fundam. Inform.* 1998, vol. 35, pp. 35–50. Available from DOI: 10.3233/FI-1998-35123403.
32. EMMERICH, Michael T. M.; DEUTZ, André H. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing*. 2018, vol. 17, no. 3, pp. 585–609. ISSN 1572-9796. Available from DOI: 10.1007/s11047-018-9685-y.
33. NGATCHOU, Patrick; ZAREI, Anahita; EL-SHARKAWI, A. Pareto multi objective optimization. In: *Proceedings of the 13th international conference on, intelligent systems application to power systems*. IEEE, 2005, pp. 84–91.
34. RAQUEL, Carlo R; NAVAL JR, Prospero C. An effective use of crowding distance in multiobjective particle swarm optimization. In: *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation*. 2005, pp. 257–264.
35. SARENI, Bruno; REGNIER, Jérémi; ROBOAM, Xavier. Recombination and Self-Adaptation in Multi-objective Genetic Algorithms. In: LIARDET, Pierre; COLLET, Pierre; FONLUPT, Cyril; LUTTON, Evelyne; SCHOENAUER, Marc (eds.). *Artificial Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 115–126. ISBN 978-3-540-24621-3.
36. ZITZLER, Eckart. Evolutionary algorithms for multiobjective optimization: methods and applications. In: 1999. Available also from: https://api.semanticscholar.org/CorpusID:14157879.

37. BINH, T. T. A multiobjective evolutionary algorithm: The study cases. In: *Proceedings of The 1999 Genetic and Evolutionary Computation Conference*. Orlando, Fla.: International Society for Genetic Algorithms, Inc. and Genetic Programming Conferences Inc., 1999.
38. FONSECA, Carlos M.; FLEMING, Peter John. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation. *IEEE Trans. Syst. Man Cybern. Part A*. 1998, vol. 28, pp. 26–37. Available also from: https://api.semanticscholar.org/CorpusID:2522644.
39. RUBIO, Pablo Torres. *Genetic Algorithms to Schedule Observations in Remote Sensing Missions*. 2024. Work in the same context and frame as the author's master's thesis.
40. EUROPEAN SPACE AGENCY (ESA). *JUICE SOC Event Catalogue* [https://juicesoc.esac.esa.int/event_tool/event/event_catalogue_5_1_150lb_23_1_b2]. 2024. Available also from: https://juicesoc.esac.esa.int/event_tool/event/event_catalogue_5_1_150lb_23_1_b2. Accessed: 2024-09-19.