POLITECNICO
MILANO 1863

DIPARTIMENTO DI SCIENZE E TECNOLOGIE AEROSPAZIALI
Master of Science in Space Engineering

Design and optimisation of an active debris removal service for large satellite constellations

Candidate:
Alessandro Maria Masserini
Matricola 883251

Supervisor:
Prof. Camilla Colombo

Co-Supervisor:
PhD candidate Simeng Huang

Academic Year 2019–2020

Alla mia famiglia

Abstract

Nowadays, access to Space is becoming more and more required since many services are offered from space missions. If this represent a great achievement made by space science, it also presents new challenges that have to be addressed. Among these, space debris play a major role. Indeed, they might affect the entire cycle of a space mission causing unexpected collisions and damages to the operational spacecraft. Together with left in orbit parts and boosters defunct satellites are the major source of space debris. Furthermore, the new rise of large constellations could lead to a rapid increase of the number of object in space. Thus, it is of utmost importance to develop strategies and missions able to remove the out-of-service satellites from large constellation missions.
This thesis work is aimed to elaborate and compare two possible types of Active Debris Removal solutions for large constellations systems to prevent the generation of massive number of debris. The study was conducted in collaboration with *D-Orbit*, a new space company founded in 2011, and *Politecnico di Milano*, participating in a program as consortium to develop an Active Debris Removal (ADR) service for large constellations. In this work, two possible mission architectures, exploiting one or more type of spacecraft will be introduced. The first solution analysed emploies a mother ship carrying de-orbiting kits, which operates attaching them to the failed spacecraft of the constellation. Once a kit is attached to the defunct satellite it will de-orbit the assembly towards an atmospheric re-entry. The second option analysed concerns the use of a single chaser, reaching all the targets and de-orbiting them one by one. For the sake of simplicity, chemical propulsion and impulsive manoeuvres are considered for the analysis. Both architectures have been studied by adopting different approaches.
In the first part of this work a mission analysis is conducted to verify the first architecture feasibility in terms of Δv and total mission time, while an optimal design strategy is developed in the second part of the thesis in the framework of the Travelling Salesman Problem. Results demonstrate the feasibility of the mission architectures considered and the possibility of reducing the amount of propellant required and the estimated mission time.

Sommario

Al giorno d'oggi l'accesso allo spazio sta diventando sempre più richiesto dato che molti servizi vengono offerti da missioni spaziali. Esso rappresenta un grande passo avanti per la ricerca spaziale ma comporta anche delle nuove sfide che devono essere affrontate. Tra tutte, quella riguardante la riduzione dei rifiuti spaziali gioca un ruolo fondamentale. Essi, infatti, possono condizionare l'intero ciclo di una missione spaziale, causando collisioni accidentali che possono danneggiare i satelliti operativi. Le maggiori sorgenti di rifiuti spaziali sono rappresentate da stadi esauriti e sgangiati dai lanciatori, parti rimaste in orbita dopo il termine di attività extra veicolari e satelliti dismessi. Inoltre, la nascita di nuove mega costellazioni può contribuire enormemente alla generazione di nuovi detriti. Lo sviluppo di startegie e missioni atte alla rimozione di satelliti non più operativi all'interno delle costellazioni è quindi di vitale importanza.
Lo scopo di questa tesi è quello di elaborare e comparare due possibili tipologie di soluzioni per la rimozione attiva dei detriti (Active Debris Removal service). Questo studio è stato condotto in collaborazione con *D-Orbit*, un'azienda spaziale fondata nel 2011, e il Politecnico di Milano, per sviluppare un servizio atti per la rimozione dei detriti spaziali. All'interno di questo lavoro, sono analizzate due possibili architetture di missione, che sfruttano l'utilizzo di uno o più tipi di veicoli spaziali. La prima soluzione presa in considerazione considera l'utilizzo di un veicolo madre equipaggiato con dei kit capaci di de-orbitare i satelliti ai quali vengono agganciati. Una volta che il kit viene agganciato al satellite non più operativo, lo guida verso un rientro in atmosfera. La seconda opzione analizzata riguarda l'utilizzo di un solo satellite, capace di raggiungere tutti gli oggetti non più in funzione all'interno di una costellazione e di guidarli verso un rientro atmosferico. Dato che si tratta di analisi preliminari, viene considerato l'uso di propellenti chimici e le manovre sono assunte come impulsive.
Le due proposte di missione sono state studiate con due metodi differenti. Le fattibilità della prima architettura è stata studiata attraverso un'analisi di missione finalizzata al calcolo del Δv e del Δt necessari, mentre il design della seconda proposta è stato ottimizzato attraverso lo sviluppo di un algoritmo capace di risolvere il Problema del Commesso Viaggiatore. I risultati dimostrano la fattibilità delle due architetture di missione proposte e l'utilità di un'ottimizzazione, con lo scopo di ridurre la quantità di carburante richiesta e il tempo totale di missione stimato.

Contents

List of Figures

List of Tables

Nomenclature

$\mathcal{P}$	Orbital period
μ	Earth's gravitational constant
ν	True anomaly
Ω	Right ascension of the ascending node
a	Semi-major axis
E	Eccentric anomaly
e	Eccentricity
J_2	Earth's first zonal harmonic
n	Mean motion
$R_\oplus$	Earth's radius
u	Argument of latitude
w	Argument of perigee

Chapter 1

Introduction

1.1 Background

A satellite constellation is a group of spacecraft with the same task, distributed in one or more orbits around the Earth. Usually, constellations are composed by a certain number of satellites, trying to cover as much terrestrial surface as possible. They are widely used, especially in the telecommunications field. One of the most important constellation is the *Global Positioning System* (GPS). It is constantly used around the world for different applications: to get position information, to drive an airplane's autopilot or simply to find something that has been lost or stolen. Every day, everyone makes use of a device that takes advantage of the presence of a satellite or a constellation in space. Internet, Earth's observation, Internet of Things (IoT) and telecommunications are the most important fields served by constellations. In the recent years, a new generation of constellations containing hundreds of small satellites, is on the way. New companies, like *SpaceX* and *OneWeb*, are going to create such large constellations, to spread the internet connection around the world. These type of constellations will fly in *Low Earth Orbit* (LEO), with all the pro and cons of such a position. Releasing a satellite in LEO requires less fuel than a geostationary one, but it is supposed to be more affected by atmospheric perturbation. Moreover, due to the less distance the coverage is reduced and a larger amount of satellites is required to achieve a full coverage of the Earth. Space companies are now pushing towards the development of smaller and cheapest satellites with respect to the past. This could lead to lower manufacturing costs and change the deployment strategy too. Indeed, dozens of this type of relatively cheap and lightweight satellites can be placed in orbit by a single launcher. However, since many satellites are needed, an efficient End-of-Life (EoL) strategy for them is mandatory, otherwise, they are going to become debris after the end of their lifetime.

1.2 State of the art

The two best known large constellations that are being built are *Starlink* and *OneWeb*. The first is going to be composed by multiple shells at different altitudes between 550 and 1300 km [10], while the second has the goal to reach a full coverage of the planet using 12 orbital planes inclined at 87.9° [9]. Up to now, *Starlink* has 240 launched satellites and the expectation is to have an half operative constellation before next five years [10]. The other competitor, *OneWeb*, has

74 satellites operating in space, before experiencing financial difficulties in the first quarter of 2020 [9].

1.2.1 End of Life techniques

The EoL strategies of *Starlink* and *OneWeb* are similar, based on the controlled re-entry, burning the satellite in atmosphere. This technique is valid only if the satellite is not out of control. If the satellite is broken, it must be considered as an uncontrolled object in orbit. The rise of large constellations could increase significantly the collision risk in LEO. Today, the band extending from 600 to 1000km above the ground is the most populated by space debris [7].

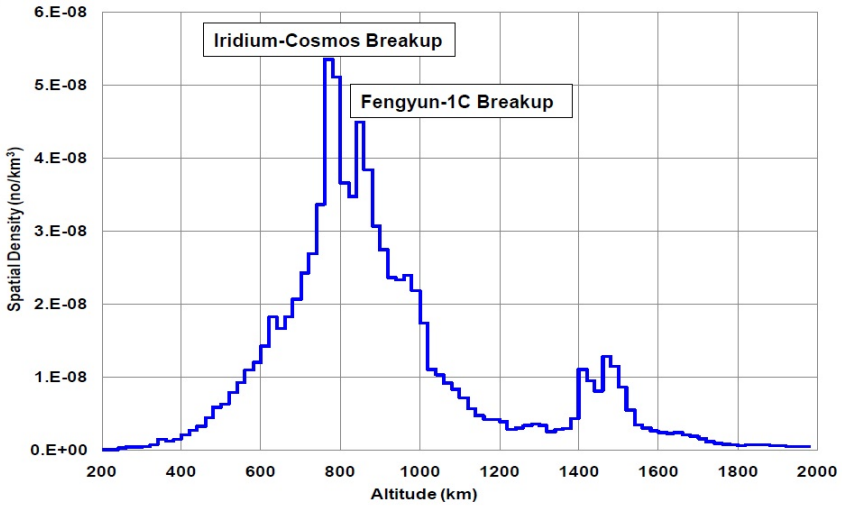


Figure 1.1: Spatial density of LEO space debris by altitude, according to a NASA report to the *United Nations Office for Outer Space Affairs*, 2011[7]

Rocket upper stages and dead spacecraft are the major sources of debris in space. According to the international regulation, the post mission orbital decay of launched objects shall be less than 25 years. This is the most important requirement concerning disposal guideline for new satellites. Although the 25 years rule is respected during the mission design, satellites may become unresponsive or uncontrollable during operation. In this case, an *Active Debris Removal* (ADR) service is required to prevent the growth of dead objects in space. Companies like *Astroscale* are developing future ADR services, trying to mitigate the problem of space debris [1]. It is of utmost importance to reduce as much as possible the number of dead objects around the Earth. Indeed, they can damage operative satellites and create a cascade effect, named *Kessler effect*. Donald Kessler, a NASA space debris expert, proposed a theoretical scenario in which the density of the objects in LEO is high enough to increase the collision risk critically and create an uncontrolled amount of space debris [5]. Besides, a chain reaction will be triggered once collision happens to a constellation, posing a severe collision hazard to the space environment. An end of life service called **ELSA-d** has been developed by *Astroscale* as a demonstrative mission with the aim to demonstrate the core technologies necessary for debris docking and removal [4].

1.2.2 Travelling Salesman Problem

When the number of space debris to be removed by a single mission increases, an optimal mission design is required to remove all the objects in the most cheapest way making the service economically sustainable. In this thesis, the travelling salesman approach will be used to design an optimal ADR mission. The state of the art concerning the solution of the TSP includes exact and heuristic algorithms. The simplest way to compute an exact solution is the so called *Brute Force* method. All the possible branches of the tree are calculated and compared to find the best one. This method always leads to the correct solution, but it is feasible only when the dimension of the problem is small. Indeed, it becomes impractical even for only 20 cities. Although this approach seems useless, it can be exploited to test the performances of heuristic and approximation algorithms. Comparing the results between brute force algorithm and a trial approximation or heuristic approach for small values of n, it is possible to determine how the trial methods will work when the dimension of the problem increases. Nowadays, high level algorithms have been developed to solve the TSP. The best performing complete algorithms are the *Branch and cut* methods, which are based on solving a series of linear programming relaxations of an integer programming problem [6]. Branch and cut algorithms belong to the family of *Branch and bound* methods and are exploited to solve *Integer Linear Programming*, where the unknown variables are constrained to be integer values. These routines exploit the *cutting planes* technique to tighten the linear programming relaxations. Indeed, the cutting plane approach allows to refine the set of variables close to the optimal ones to minimise the objective function. Usually, when the dimension of the problem is very large, another useful approach is to split the problem into sub-problems, solving them separately. The state of the art branch and cut methods are able to solve the TSPs characterised by 1000 to 3000 vertices within modest computational times [6]. For larger problems the computational time increases significantly. Currently, the largest TSP solved is instance $d15112$ having 15112 vertices. The known optimal solution for this instance was calculated in September 2003. Although these impressive results, complete algorithms suffer from some limitations. The first one is that computational times becomes rapidly prohibitive, as the instance size increases. Introducing the concept of CPU time it is possible to have a clear idea on the performance of an algorithm. The CPU time measures the time a CPU takes to elaborate a specific operation. It is expressed in clock cycles and converted in seconds knowing the CPU operational frequency, and it does not include the time spent to run input-output (I/O) operations. Exploiting a single CPU it is always lower with respect to the real elapsed time, since this one includes also I/O operations. Otherwise, using 2 CPU parallelly taking 2 seconds each to run a program the total CPU time is equal to 4 seconds, while the real elapsed time is 2 seconds since the two processors are working together. Instance $d15112$ was solved by a network running 110 workstations, but the computational time on a *Compaq EV6 Alpha processor* running at 500MHz CPU is estimated to be 22.6 CPU years. Secondly, computational time is not only dependent on the number of vertices of the graph but depends also on the type of instance. For example, instance $pr2392$ with 2392 vertices required a computational time of 116 CPU seconds on a 500MHz *Compaq XP1000* workstation, while instance $d2103$ with 2103 vertices was solved using a network of 55 *Alpha 21164* processors running at 400 and 500MHz. The estimated total run-time on the same test bench of instance $pr2392$

was about 129 CPU days. Given the limitations of complete algorithms, there is a large interest in others methods to face the TSP. *Stochastic local search* (SLS) and heuristic algorithms are widely used today to solve optimisation problems. The great advantage of these methods is the possibility to set a trade-off between the total computational time and the quality of the solution obtained. Problems that are characterised by a family of good solutions closer to the optimal one, can be solved rapidly without converging exactly to the lowest possible value, but to an acceptable close one. SLS algorithms are able to reach optimal or near-optimal solutions for symmetric graphs with thousands of cities with seconds or minutes of CPU times on modern workstations [6]. For example, instance $d15112$ was solved using a 500MHz *Alpha* processor in about seven hours of CPU time finding a solution with an error of 0.0186% with respect to the known optimum. Other types of algorithms suitable to solve TSP are based on iterative methods, like genetic algorithms. They require the choice of an initial guess or population and the performances are strongly dependent on the nature of the graph analysed. It has been demonstrated that the asymmetric graphs are more difficult and they require more computational time to solve, no matter which type of algorithm is used [6]. A graph is called asymmetric when there is at least one pair $\{j, k\}$for which $w(j, k) \neq w(k, j)$, being $w(j, k)$ the cost to go from $\{j\}$ to $\{k\}$ and vice versa. As it will be shown in this chapter, the set of failed satellites to be reached inside a constellation can be modelled as an asymmetric graph. Solving asymmetric TSP (ATSP) with large number of vertices could become unfeasible also for the most efficient complete algorithm in terms of computational time. In this case approximation methods are preferred. One of the most famous approximation routine is called *Nearest Neighbour Algorithm* (NNA). NNA solves the TSP starting from a random city and always visiting the nearest until all cities are reached. In order to have a more precise result, within an instance including n cities, NNA can be launched n times with one city selected as the starting point at each time. In this way, among all n results, the best one can be found. It was demonstrated that NNA rarely converged to the optimal solution.

1.3 Scope of the thesis

Designing an efficient ADR service is a challenging work, it has to be both reliable and economically sustainable. The study was conducted in collaboration with *D-Orbit*, a new space company founded in 2011, and *Politecnico di Milano*, participating in a program as consortium to develop an *Active Debris Removal* (ADR) service for large constellations. The aim of this work of thesis is to perform the mission analyses for two different mission architectures, in two respective ways. The framework of *architecture 1* exploits the use of a mother ship filled with kits to deorbit a set of failed satellites. Typical engineering requirements for space missions will be taken into consideration, to study the feasibility of the this architecture. Within the context of *architecture 2* the operations conducted by a chaser will be optimised to reduce as much as possible the total mission time. The goal of the study is to define a set of rules, to optimise the sequence in which dead satellites within a constellation are to be reached, to minimise the total time of flight. This optimisation problem will be solved within the context of the *Travelling Salesman Problem* (TSP). The idea is to develop an algorithm capable to identify the best sequence to reach dead spacecraft inside a given constellation. This could be a starting

point to those companies developing ADR services for future large constellations.

1.4 Thesis structure

The remaining of this thesis is organised as follows:

- Chapter 2 gives a description and analysis about the orbital manoeuvres involved, focusing on their optimisation. Orbital perturbations exploited to design the manoeuvres are presented and analysed. In this chapter, the two mission architectures proposed are described too, highlighting both differences and common points between the set of orbital transfers required by these two architectures.

- Chapter 3 presents the analysis and simulations concerning *architecture 1*, given prescribed requirements. Results of this study are presented and discussed.

- Chapter 4 is divided into two parts. The first one is devoted to the presentation and description of the *Graph theory*, the *Travelling Salesman Problem* and how to apply it to the optimisation of an ADR service. The second part of the chapter is devoted to the development of algorithms capable of solving the TSP applied to *architecture 2*. Results are discussed at the end of the chapter.

Chapter 2

Mission architectures and manoeuvres design

This chapter will introduce two different mission architectures that have the application potential for ADR services, following which, the design of the orbital manoeuvres that are involved in the two architectures will be presented.

2.1 Architectures definition

2.1.1 Architecture 1 - Mother ship plus kits

In this framework, two different types of vehicle are exploited. The first, named "*mother ship*", has the task of reaching each target and perform the most critical operations. Inside the *mother ship*, a set of *kits* are stored. They must be attached to each target by the *mother ship* and they are in charge of de-orbiting operations. Thus, the number of targets the *mother ship* shall be able to reach is equal to the number of onboard *kits*. It is possible to remove an extra dead satellite from the constellation considering a scenario in which the *mother ship* is able to de-orbit itself with a target attached. The main vehicle (i.e. the *mother ship*), equipped with a main thermochemical engine, is designed as a complex system, capable to execute orbit raising and transfers within the constellation, relative navigation tasks and to dock with each failed satellite. Sophisticated technologies, like a robotic arm or other mechanical systems, will be required, to make the *mother ship* able to attach the *kit* to the target once docked to it. The *kit* vehicle is provided with a smaller engine and it is tasked to handle only de-orbiting operations, moving the satellites to the disposal orbit. A detailed description of the orbital manoeuvres involved is presented in section 2.3, while the mission analysis of the architecture is presented in chapter 3. The complete operations sequence is explained here:

1. The *mother ship* is released by the launcher into a parking orbit whose altitude is lower with respect to the operational altitude of the constellation. An orbit raising manoeuvre performed by the *mother ship* brings the spacecraft into the same constellation orbit where dead satellites need to be reached. If the Right Ascension of the Ascending Node (RAAN) of the parking orbit does not match with the RAAN of the operational plane, the different nodal regression rate between the orbits induced by the effect of Zonal Harmonics is exploited to adjust the RAAN parameter, as explained in section 2.2.

2. Once the *mother ship* has completed the orbit raising phase, rendezvous operations to attach a *kit* to the first satellite begin. To handle this phase, a phasing manoeuvre is exploited in order to reach the first satellite. The design of this type of transfer is explained in section 2.3.1.

3. The next step involves attaching the *kit* to the dead spacecraft through the use of a robotic arm. Once the connection between the satellite and the *kit* is established, the *mother ship* starts to move towards the next target exploiting again a phasing manoeuvre.

4. The *kit* will exploit its engine to move the satellite into the disposal orbit. Possible collisions with satellites must be avoided during this phase. For this purpose, the apogee radius of the disposal orbit is designed to be at least 100 km below the constellation altitude.

5. The procedure is repeated until all the dead satellites have been reached.

2.1.2 Architecture 2 - Chaser

In this framework, a spacecraft named "*chaser*" is tasked with reaching and de-orbiting every single target. It is equipped with a thermochemical engine which is in charge of performing all the required manoeuvres. The maximum number of targets the *chaser* is able to reach is constrained by the amount of propellant onboard. A series of de-orbiting manoeuvres followed by subsequent orbit raising will be exploited by the *chaser* to fulfil the mission tasks. The total mission time depends on the order in which satellites are reached. An optimal design of this architecture, aimed to minimise the total Δt, is discussed in chapter 4. The complete operations sequence is explained here:

1. The *chaser* is released by the launcher into a parking orbit where the orbit raising phase starts. At the end of this phase the first satellite is reached by the *chaser*.

2. The failed satellite is moved to the disposal orbit. The apogee altitude is reduced to avoid collisions with operational satellites, while the perigee altitude is decreased to exploit drag perturbation.

3. Once the satellite is placed into the correct disposal orbit, the *chaser* will detach from it and will transfer towards the next one. If the next target lies on the same orbital plane a simple coplanar phasing transfer is used to reach the spacecraft, otherwise a change of RAAN is required before.

4. The procedure is repeated until all the spacecraft have been reached.

2.2 Secular effects from zonal harmonics

Both architectures may require an out-of-plane transfer when a satellite, which need to be reached, lies in a different orbital plane. As it is well known a plane change manoeuvre is very expensive in terms of Δv. Thus, to reduce the cost of the transfer, the idea is to let the ADR service spacecraft (*mother ship* or *chaser*) drift towards

the orbital plane of the target, by exploiting the zonal harmonics perturbations. The first-order secular effects on an Earth satellite caused by the non spherical gravity field come from the even zonal harmonics of the Earth's gravity field. In particular, only the secular effects associated with the J_2 are considered in this work, discarding the perturbations coming from other gravitational harmonics. Short period effects are responsible for the change of all the orbital elements, while secular variations due to J_2 affect only the nodal regression and the apsidal rotation. Nodal regression moves the RAAN backwards with respect to its original value. RAAN is measured eastward starting from the γ direction (First point of Aries) to the node. It means that the orbit is affected by a counter rotation around the polar axis. Moreover, J_2 is responsible also for the secular effect of the apsidal rotation. Equations 2.1 and 2.2 refer to the secular variations of the right ascension of the ascending node and the argument of perigee.

$$\dot{\Omega}_{sec} = -\frac{3nR_{\oplus}^2 J_2}{2p^2} cos(i) \tag{2.1}$$

$$\dot{w}_{sec} = \frac{3nR_{\oplus}^2 J_2}{4p^2} \left[4 - 5sin^2(i)\right] \tag{2.2}$$

where n is the mean motion, $R_{\oplus}$ the Earth's radius, p the semi-parameter defined as $p = a(1 - e^2)$ and i the orbit inclination. These effects have been considered in this work to design cheaper out-of-plane transfers, in order to allow a spacecraft to reach a target on a separated orbital plane without the use of large amount of propellant. Indeed, it is of utmost importance to save as much fuel as possible to move the ADR service spacecraft (*mother ship* or *chaser*) from one target to the next, maximising the number of satellites de-orbited per mission. In section 2.3.3 out-of-plane transfers exploiting J_2 perturbing effect are presented.

2.3 Orbital manoeuvres

The set of orbital manoeuvres involved for each architecture is different. Both *architecture 1* and *architecture 2* requires the design of in-plane and out-of-plane transfer to accomplish the mission. The former category refers to the manoeuvres that change only the in-plane orbital elements, i.e. the semi-major axis, the eccentricity, and the argument of latitude. The latter category refers to the manoeuvres that change the inclination. Note that in this thesis the RAAN is always changed by exploiting the J_2 effects for the purpose of saving propellant.In this section the two sets of manoeuvres concerning *architecture 1* and *architecture 2* are presented and discussed.

2.3.1 Circular coplanar phasing

In the framework of *architecture 1* the *mother ship* is supposed to move from a target to the next on the same orbit by using a circular coplanar phasing. Indeed, it is possible to exploit this transfer to reach a target on the same orbit with a different argument of latitude u, as described in [3]. Two types of transfers are possible, depending if the target is leading or trailing the *mother ship*. A phase angle θ is defined measured from the target to the interceptor. At this point, the strategy is

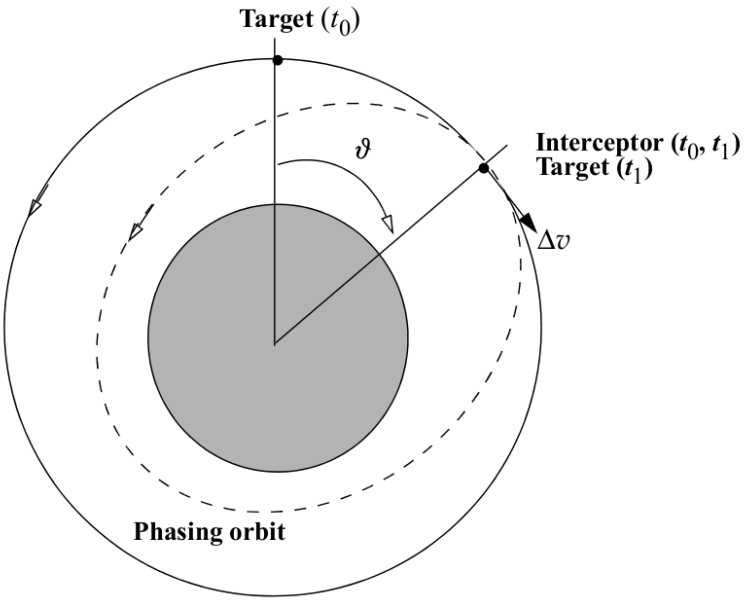


Figure 2.1: Target leading the interceptor, i.e the *mother ship* [3]

to move the interceptor to an elliptical phasing orbit with a period τ_{phase} equal to:

$$\tau_{phase} = \frac{k_{tgt}(2\pi) + \theta}{w_{tgt}} \tag{2.3}$$

where k_{tgt} is the number of revolutions of the target satellite and w_{tgt} is the target angular velocity, identified on the circular orbit as the mean motion n:

$$w_{tgt} = \sqrt{\frac{\mu}{a_{tgt}^3}} = n \tag{2.4}$$

with μ gravitational planetary constant and a_{tgt} semi-major axis (i.e. the radius on the circular orbit). While the interceptor completes a revolution on the phasing orbit the target completes a revolution minus the phase angle θ.
The the semi-major axis of the phasing orbit a_{phase} is strictly related with the phasing time through the number of revolutions of the interceptor k_{int}:

$$a_{phase} = \left(\mu \left(\frac{\tau_{phase}}{k_{int} 2\pi} \right)^2 \right)^{\frac{1}{3}} \tag{2.5}$$

The sum of the two tangential burns to enter and escape from the phasing orbit represents the total Δv required (eq2.6).

$$\Delta v = 2 \left| \sqrt{\frac{2\mu}{a_{tgt}} - \frac{\mu}{a_{phase}}} - \sqrt{\frac{\mu}{a_{tgt}}} \right| \tag{2.6}$$

Formula 2.3 relates the time of flight with the number of revolution of the target, while equation 2.6 relates the Δv with the semi-major axis of the target. It is well known that the period of an orbit increases along with the increase of the value of the semi-major axis, while the Δv required increases as the difference between the

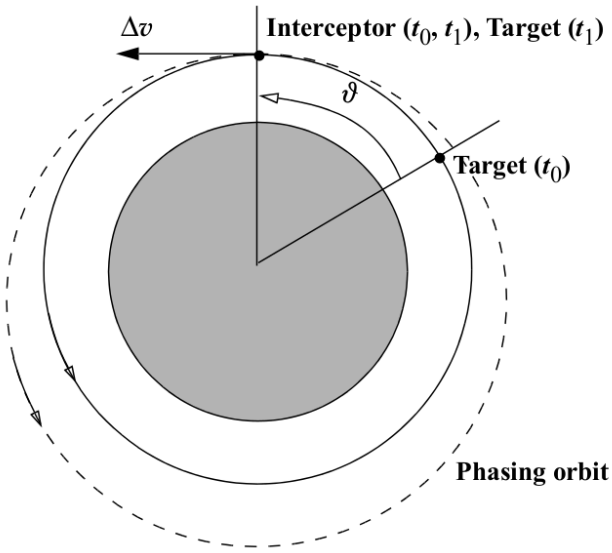


Figure 2.2: Target trailing interceptor, i.e the *mother ship* [3]

semi-major axis of the phasing orbit and the starting orbit becomes bigger. This behaviour suggests that it is not possible to optimise both time of flight and Δv at the same time because when the first increases the second will decrease and vice versa. In the following analysis a circular coplanar phasing between two satellites is described. The orbital parameters referring to the following example are:

Orbital parameter	Interceptor	Target
Semi-major axis [km]	1200	1200
Eccentricity	0	0
Argument of latitude [degrees]	60	90

The number of revolutions for both the target and the interceptor is fixed to be equal to 1. The possible optimal transfer can be theoretically found imposing Δv and Δt as optimisation targets and k_{tgt}, k_{int} as optimisation variables. The constraints of the multi-objective optimisation are possible limit values on Δv and Δt, and the radius of perigee of the phasing orbit, that should be kept larger than the Earth's radius plus a contingency altitude. Although this could appear a fancy solution, it should be kept into consideration that it will depends on the relative importance between Δv and Δt, that could change depending on the situation. To reduce the complexity of the problem the equality $k_{tgt} = k_{int} = k$ can be imposed. In this situation the interceptor and the target performs the same number of revolutions to achieve the phasing. The set of solutions for the transfer is shown in figure 2.3. All the possible solutions lie in a pareto front generated running the simulation for different values of k.

2.3.2 Coplanar phasing between different orbits

In the framework of *architecture 2* this transfer is required to move the *chaser* from the disposal orbit to the next target. The *chaser* is supposed to move from an internal elliptical orbit directly to the target on the outer circular orbit. The idea is derived from the circular coplanar case, trying to exploit a phasing orbit to start

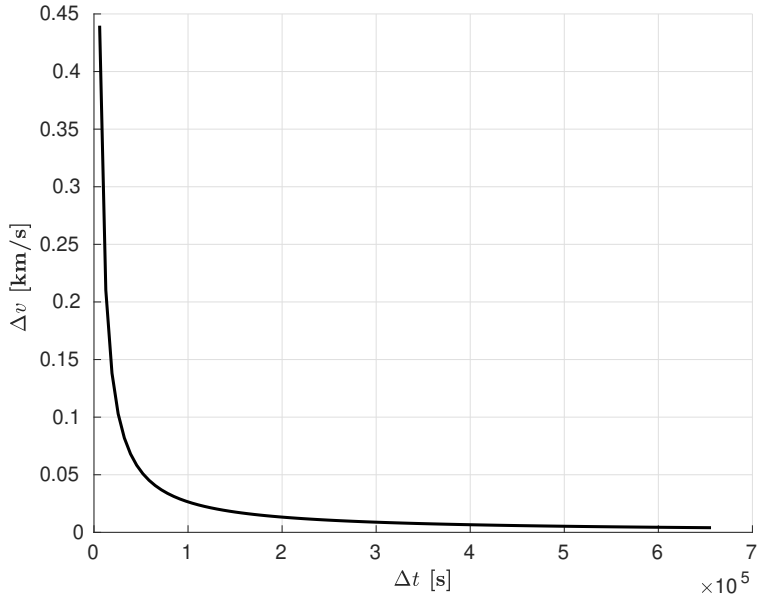


Figure 2.3: Δv vs Δt for varying number of revolutions $k = 1, 2, ..., 100$

the engines at the exact time instant to make the interceptor and the target meet on the final orbit. The transfer is shown in figure 2.4.

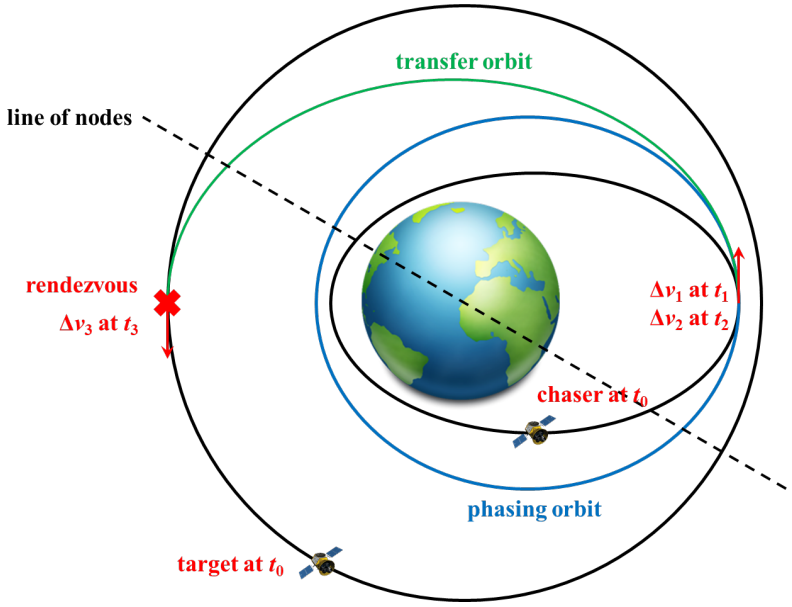


Figure 2.4: Coplanar phasing between different orbits

To make the transfer as cheapest as possible a bi-tangent manoeuvre has to be designed. This can be achieved starting from the apogee or from the perigee of the elliptical orbit, in order to have only a tangential variation of the velocity vector. Once the *chaser* has reached one of the two apsidal points, it moves into an intermediate orbit, waiting for the right position of the target to begin the transfer. In this section it will be analysed the case where the manoeuvre is performed at the apogee of the elliptical orbit. First of all, the time the *chaser* takes to reach the

apogee has to be calculated. Defining the eccentric anomaly E:

$$sin(E) = \frac{sin(\nu)\sqrt{1-e^2}}{1+ecos(\nu)} \tag{2.7}$$

$$cos(E) = \frac{e+cos(\nu)}{1+ecos(\nu)} \tag{2.8}$$

being e and ν the eccentricity and the true anomaly of the orbit. It is now possible to calculate the time to reach the apogee through the Kepler's equation:

$$t - t_0 = \frac{1}{n}\left[2k\pi + E - esin(E) - (E_0 - esin(E_0))\right] \tag{2.9}$$

with $k = 0$ additional revolutions of the *chaser* are discarded. To define the waiting orbit, the required position of the target at the beginning of the transfer should be computed. Target and *chaser* will meet at $u_{tgt} = w_{int}$ on the circular orbit, where w_{int} is the argument of perigee of the elliptical orbit. The angle spanned by the target during the transfer is then:

$$\alpha = \mathcal{P}_{tr} n_{tgt} \tag{2.10}$$

where $\mathcal{P}_{tr}$ is the period of the transfer orbit and n_{tgt} the mean motion of the target. The argument of latitude of the target at the beginning of the transfer is given as:

$$u_{tgt} = w_{int} - \alpha \tag{2.11}$$

The calculation of the semi-major axis of the phasing orbit a_{wait} is the last step. In order to not waste energy with respect to a direct transfer from the apogee to the circular orbit the apogee radius of the phasing orbit $r_{a\ wait}$ need to be less than the target orbit radius R_c and greater than the perigee radius of the *chaser* orbit r_p.

$$a_{wait} = \left(\mu\left(\frac{t_{wait}}{2\pi}\right)^2\right)^{\frac{1}{3}} \tag{2.12}$$

with t_{wait} equal to:

$$t_{wait} = \frac{u_{tgt} - u_{tgt0}}{n_{tgt}} \tag{2.13}$$

where u_{tgt0} is the position of the target at the time the *chaser* reaches the apogee. If the constraint of equation 2.15 is not satisfied the whole procedure is repeated at the next apogee passage.

$$r_{a\ wait} = 2a_{wait} - r_p \tag{2.14}$$

$$r_p < r_{a\ wait} < R_c \tag{2.15}$$

The total cost of the transfer is calculated as the sum of these three Δv:

$$\Delta v_1 = v_{a\ wait} - v_{a\ int} \tag{2.16}$$

$$\Delta v_2 = v_{p\ phase} - v_{a\ wait} \tag{2.17}$$

$$\Delta v_3 = v_{circ} - v_{a\ phase} \tag{2.18}$$

$$\Delta v = |\Delta v_1| + |\Delta v_2| + |\Delta v_3| \tag{2.19}$$

being $v_{a\ wait}$ the velocity at the apogee of the phasing orbit, $v_{a\ int}$ the velocity at the apogee of the *chaser* orbit, $v_{p\ phase}$ the velocity at the perigee of the transfer arc and v_{circ} the target velocity on the circular orbit.

2.3.3 Out-of-plane manoeuvres

Two types of out-of-plane manoeuvres are investigated in this work. The first one is the inclination change manoeuvre between circular orbits, while the second one is the change in RAAN. In the framework of *architecture 1* the inclination change is exploited to trigger a change of RAAN and to adjust the orbital inclination after the orbit raising. It is a relative expensive manoeuvre devoted to the rotation of the velocity vector of the spacecraft. Indeed, no gain in the velocity modulus is obtained because the Δv provided is aimed to change only the direction of the velocity vector. The Δv required is:

$$\Delta v = 2 v_{circ} sin(\Delta i) \tag{2.20}$$

where v_{circ} is the velocity on the circular orbit and Δi is the change in inclination. In order to not lead to an undesired change of RAAN, the manoeuvre must be performed in correspondence of the nodes. It is always better to perform this type of manoeuvre when the tangential velocity is lower, to save as much fuel as possible. Thus, this manoeuvre is performed at the end of the orbit raising, when the altitude is increased and the velocity is decreased. Moreover, it can be exploited to modify the nodal regression rate of the orbit changing the inclination. This will allow the spacecraft to drift towards another orbital plane without the need to change the RAAN with a single engine burn. Indeed, a small instantaneous variation of the inclination value will lead to a big variation in terms of RAAN, due to the effects of zonal harmonics perturbations. This strategy has been adopted within the context of . *architecture 1.* Whenever the *mother ship* is required to move from a constellation plane to another, a small variation of the inclination value is provided to trigger the RAAN change. Thus, the change in RAAN manoeuvre for *architecture 1* can be summarised in three steps:

1. The inclination of the *mother ship* orbit is decreased with an inclination change manoeuvre through an impulsive burn. Thus, a nodal regression rate different with respect to the constellation planes is obtained.

2. The *mother ship* is let to drift freely towards the next operational plane exploiting the J_2 effect. The time it takes to reach the next plane is:

$$\Delta t = \frac{\Omega_{0int} - \Omega_{0tgt}}{\dot{\Omega}_{tgt} - \dot{\Omega}_{int}} \tag{2.21}$$

   where Ω_{0int}, $\dot{\Omega}_{int}$ and Ω_{0tgt}, $\dot{\Omega}_{tgt}$ are the initial *mother ship* and target RAAN and nodal regression rates.

3. Once the *mother ship* RAAN is equal to the target one another inclination change manoeuvre is performed to adjust the inclination of the *mother ship* orbit. The total Δv required for the transfer is then two times the one required by the inclination change manoeuvre:

$$\Delta v = 2 \left(2 v_{circ} sin(\Delta i)\right) \tag{2.22}$$

In the framework of *architecture 2* no inclination change manoeuvre are required to trigger the RAAN change. Indeed, the *chaser* is supposed to move from the disposal orbit directly towards the next operational plane. Since the disposal orbit

is elliptical and it is characterised by a lower semi-major axis with respect to the constellation, it has already a different nodal regression rate compared to the target operational orbit. Thus, the *chaser* is ready to drift exploiting J_2 effect immediately after the disposal operations. The change in RAAN manoeuvre steps for *architecture 2* are the followings:

1. The *chaser* performs the disposal manoeuvre and brings the first satellite to the disposal orbit. At the apogee of the disposal orbit it detaches from the dead spacecraft and starts the engine to raise its perigee altitude in order to not be affected by drag effect during the drift time.

2. The *chaser* is let to drift freely towards the next operational plane exploiting only J_2 effect.

3. When the correct RAAN is reached a coplanar phasing is performed to match the next satellite on the operational orbit.

Since the *chaser* is let to drift on an elliptical orbit the apsidal rotation shall be taken into consideration too.

2.3.4 De-orbit manoeuvre

The de-orbiting operations are identical for both architectures. The adopted strategy is aimed to move the target into an elliptical orbit and to exploit drag perturbations, driving the satellite towards an atmospheric re-entry. Collisions with other operational satellites shall be avoided. A bi-tangent transfer is adopted to decrease the perigee and the apogee of the operational orbit. The perigee altitude is calculated taking into consideration the area to mass ratio of the satellite to exploit drag effects, while the apogee altitude is fixed to be 100 km lower than the constellation one for collision avoidance. In the framework of *architecture 1* these operations are in charge of the *kit* only, while within the context of *architecture 2* the *chaser* is tasked of performing the disposal transfer. The complete sequence is explained here:

1. The *kit* (or the *chaser*) performs an impulsive manoeuvre to decrease the perigee altitude.

2. Once the *kit* (or the *chaser*) has reached the perigee of the transfer elliptical orbit a second burn is performed to decrease the apogee altitude for collision avoidance.

3. In the framework of *architecture 1* the kit and the satellite will de-orbit together, conversely the *chaser* will detach from the satellite moving towards the next one.

Chapter 3

Mission analysis for architecture 1

This chapter presents the mission analysis for the first mission architecture – *mother ship* and *kits*. The aim is to evaluate the performances for the first architecture in terms of the mission time and velocity change. Two different scenarios are considered:

1. *Active Debris Removal* **Scenario 1** (ADR1): each plane of the constellation is served by one *mother ship*.

2. *Active Debris Removal* **Scenario 2** (ADR2): each *mother ship* is in charge of serving more than one plane.

For this study the *OneWeb* constellation is taken into consideration. The orbital parameters are listed in table 3.1.

3.1 Mission requirements

The mission requirements for the *mother ship* are the followings:

- The maximum available Δv is 1km/s.

- The maximum lifetime is 2 years.

- ADR1: one *mother ship* is responsible for one plane of failed satellites.

- ADR2: one *mother ship* is responsible for multiple planes of failed satellites.

- The first failed satellite must be reached by the *mother ship* within six months upon the mission start and the service request.

Mission requirements for the *kits* are the following:

- The maximum re-entry time is 5 years, starting from the separation from the kit, and ending with a demise altitude of 78 km.

- Safety distance from internal constellation satellites for collision avoidance shall be at least 100 km.

Table 3.1: Constellation data

Prameter	Value
Altitude [km]	1200
Eccentricity	0
Inclination [degrees]	87.9
Number of planes	12
Number of spacecrafts per plane	49

In the OneWeb constellation considered in this study, all orbital planes are evenly spaced along the equatorial plane. They are assumed to be separated in altitude by a distance of 4 km for collision avoidance. In each orbital plane, satellites are also evenly distributed. A representation of the constellation is shown in figures 3.1 and 3.2.

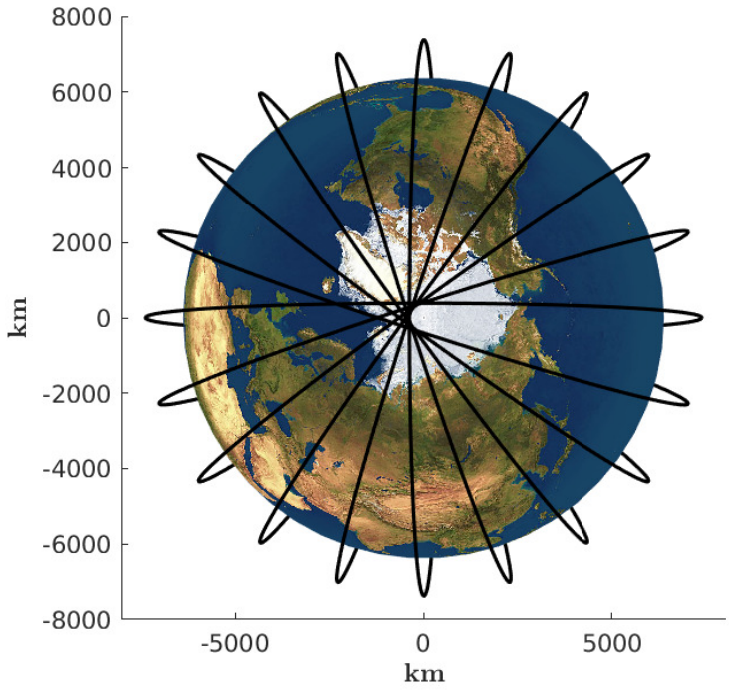


Figure 3.1: Orbital planes distribution, polar view

The release orbit for both ADR1 and ADR2 missions is assumed to have the parameters listed in table 3.2. All the *mother ships* are released into the last orbital plane such that the first service can be provided instantly without need to wait for the RAAN match. It is worthy to note that the inclination is not the same of the constellation to accelerate up the RAAN drifting rate of the *mother ships* towards their operational planes. The drift will occur by exploiting J_2 perturbations. Since the altitude and the inclination of the release orbit are lower than those of the constellation, the nodal regression of the *mother ships* will be faster than that of the constellation. In this way no manoeuvres is performed for RAAN changing.

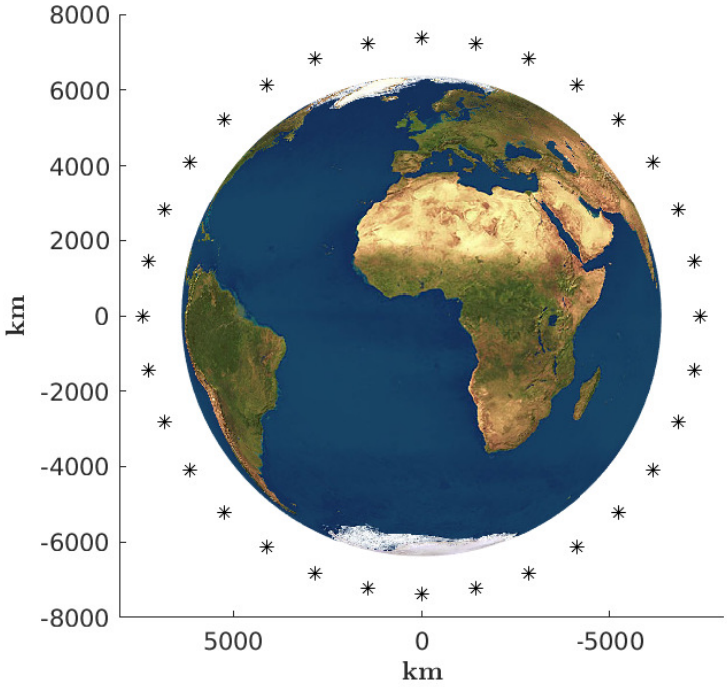


Figure 3.2: Distribution of the satillets lying on a single orbital plane

Table 3.2: Orbital parameters of the release orbit for ADR1 and ADR2 missions

Prameter	Value
Altitude [km]	500
Eccentricity	0
Inclination [degrees]	86
RAAN [degrees]	167.42

3.2 Mission analysis for ADR1

As stated in chapter 2.1.1, each *mother ship* is equipped with a set of de-orbit *kits*. A set of 8 kits is considered in this analysis. Moreover, the *mother ship* is considered to be capable to de-orbit itself with a target attached. Thus, every *mother ship* is able to de-orbit a number of satellites equal to the number of *kits* onboard plus an extra one during de-orbiting itself. To increase the robustness of the simulation, the distribution of the failed satellite in the worst case is considered. Thus, they are supposed to be equally spaced inside the plane. Input data of the simulation are listed in table 3.3. The mission phases are summarised as the followings:

- **Phase 1**: After the launch all the *mother ships* lie on the same circular parking orbit. Since ADR1 considers one *mother ship* working per each plane, 12 *mother ships* are needed for the mission. 11 of them need to drift at the altitude of 500km to reach their respective operational planes. The time that each *mother ship* will take to reach its operational plane by exploiting J_2 perturbation is shown in figure 3.3.

Table 3.3: Input data for ADR1 simulation

Prameter	Value
Number of failed satellites per plane	9
Number of kits per mother ship	8
Number of mother ships	12
Separation angle between satellites [degrees]	40

- **Phase 2**: After each *mother ship* has reached the correct RAAN, an Hohmann transfer will bring it to the altitude of the operational plane and then a plane change manoeuvre is performed to adjust the inclination to 87.9°.
- **Phase 3**: Each *mother ship* cleans its plane with a series of phasing manoeuvres.

Simulation results and discussion

After the release by the launcher each *mother ship* starts to drift towards its operational plane. The x axis of figure 3.3 represents the *mother ships* sorted by numbers from 1 to 12. The first refers to the *mother ship* serving the last orbital plane. It is possible to observe how a considerable value of time the last *mother ships* take to reach their respective planes. Indeed, since the difference of inclination between the releasing orbit and the constellation orbit is only of 1.9 degrees, there is no significant difference in the nodal regression rate. However, a bigger difference in inclination would lead to a significant utilisation of propellant when the *mother ship* will need to adjust its inclination according to constellation one. Results of the orbit raising including the Hohmann transfer and the inclination change are shown in figures 3.4 and 3.5. This phase requires less than one hour but it costs more than an half of the Δv available onboard. Increasing the altitude of the release orbit will decrease the cost of this transfer, since the energy difference between the interceptor and target orbits would become smaller. As a possible drawback of this choice, the time required by **Phase 1** will increase. Indeed, increasing the semi-major axis of the release orbit will lower its nodal regression rate. Looking at formula 2.3, it is clear that the phasing time between the interceptor and the target depends on the number of revolutions selected. In this analysis k_{int} and k_{tgt} are set to be equal and they was set in order to keep the total time of this phase lower than two months. It is now possible to calculate the residual Δv onboard available (figure 3.9) and the residual lifetime (figure 3.8) for each *mother ship*. Looking at the results, it is possible to say that the strategy adopted by ADR1 mission is capable to accomplish all the requirements. The total mission time for each *mother ship* is lower than the lifetime and a significant amount of Δv is left to de-orbit itself together with the last failed satellite.

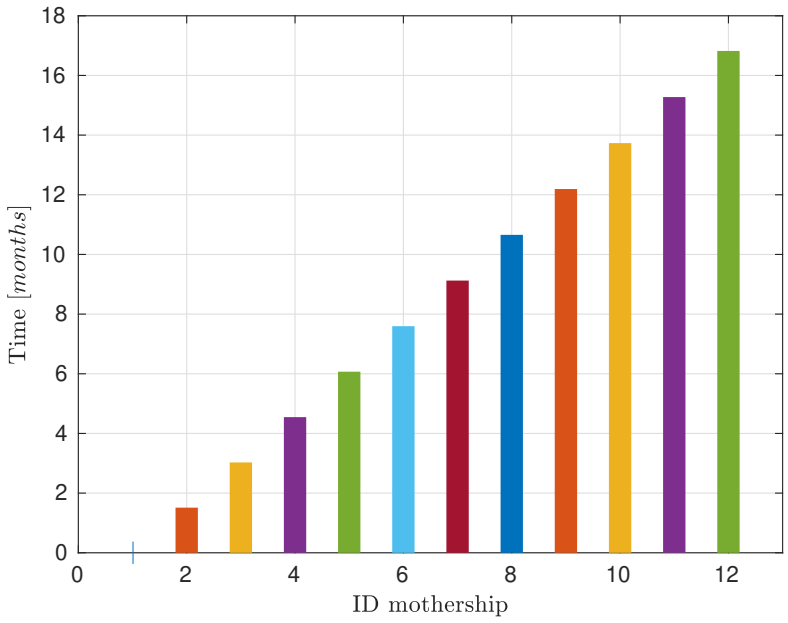


Figure 3.3: **Phase 1:** time needed for each *mother ship* to reach its operational plane

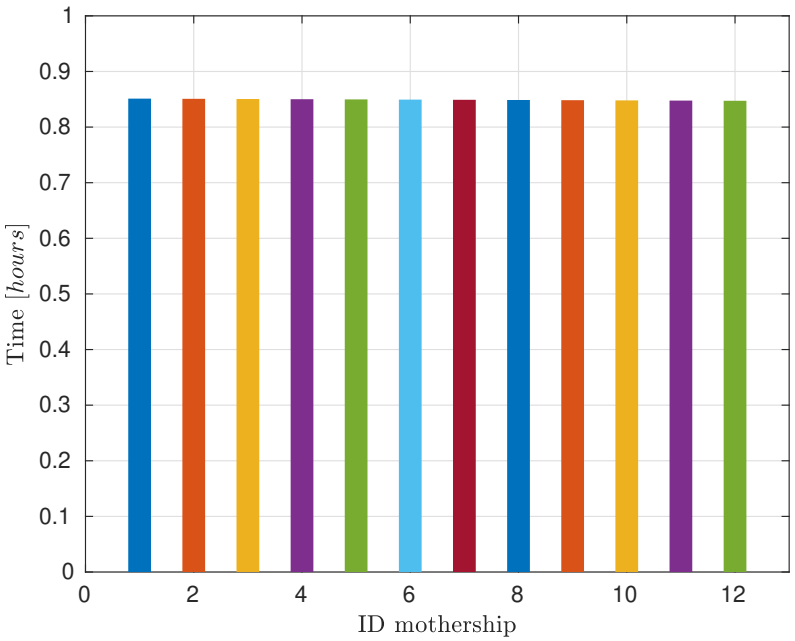


Figure 3.4: **Phase 2:** time required by the Hohmann transfer and the inclination change to complete the orbit raising

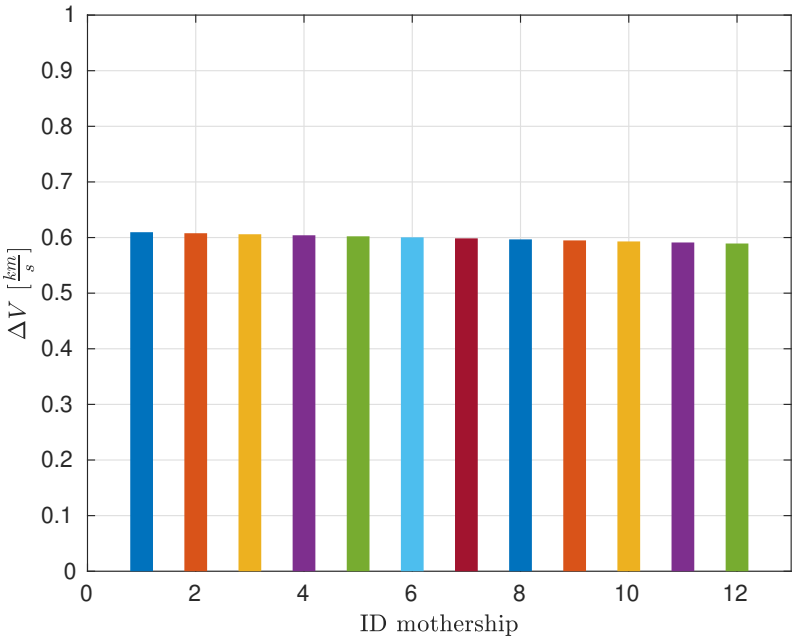


Figure 3.5: **Phase 2:** Δv required by the Hohmann transfer and the inclination change to complete the orbit raising

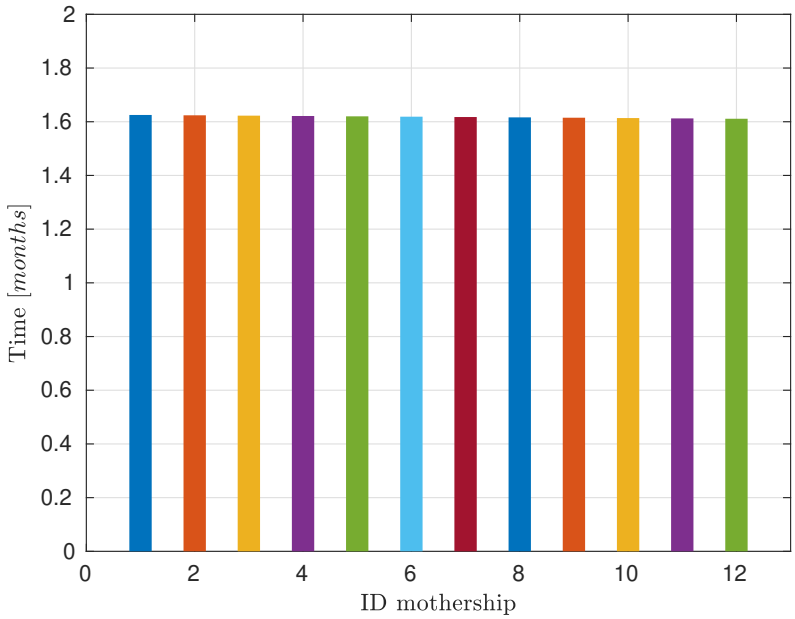


Figure 3.6: **Phase 3:** time required by each *mother ship* to reach its 9 failed satellites

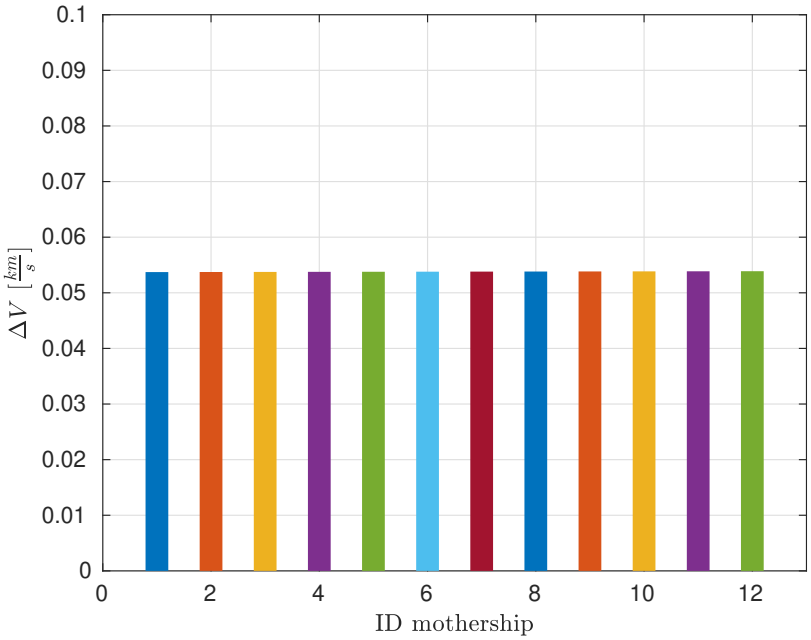


Figure 3.7: **Phase 3:** Δv required by each *mother ship* to complete the phasing manoeuvres to reach the 9 failed satellites

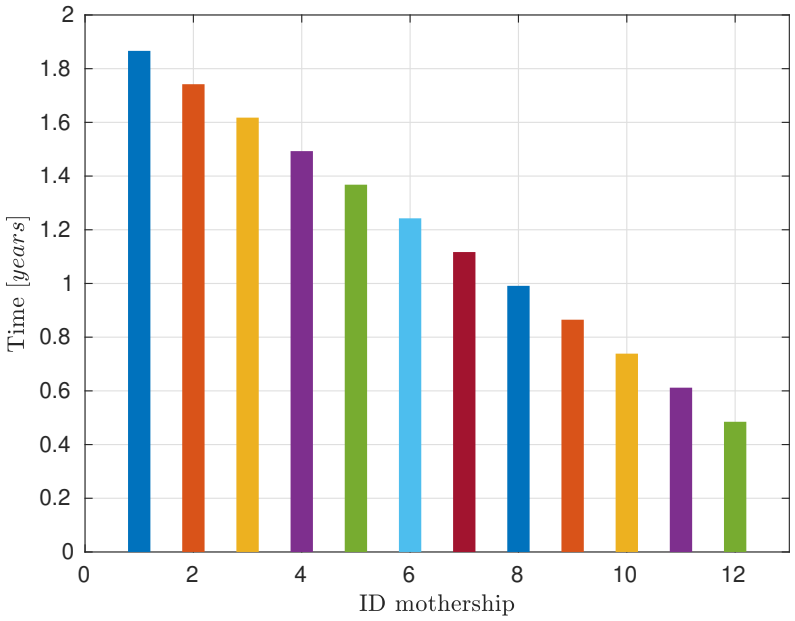


Figure 3.8: **Phase 3:** residual lifetime of each *mother ship*

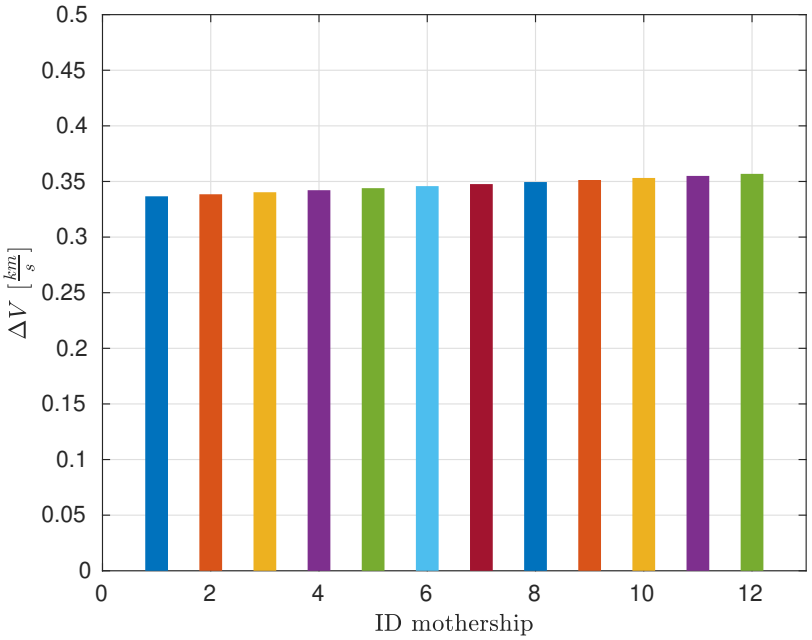


Figure 3.9: **Phase 3:** residual Δv available onboard of each *mother ship*

3.3 Mission analysis for ADR2

In ADR2 mission, one *mother ship* is in charge of multiple planes of failed satellites. Each *mother ship* is required to finish cleaning the planes of failed satellites that it is in charge of within its lifetime, given the maximum total Δv. ADR2 is an extension of ADR1. In both ADR1 and ADR2, the *mother ship* follows the same procedure from launch until the removal of all the failed satellites in the first orbital plane. However, in ADR2, after each *mother ship* has finished to clean its first plane of failed satellites, it shall drift to the next plane exploiting the J_2 effect, and then the third plane, until it runs out of the propellant and lifetime. ADR2 simulation is conducted considering a target rich environment with 4 failures along each orbital plane. A set of 8 *kits* onboard each *mother ship* is considered. Each one is in charge of cleaning two consecutive planes, drifting to the next exploiting J_2 effect once it has finished the work with the first plane. Input data for ADR2 mission are listed in table 3.4. To increase the robustness of the simulation, the distribution of the failed satellite in the worst case is considered. Thus, they are supposed to be equally spaced inside the plane. Mission phases are here summarised:

Table 3.4: Input data for ADR2 simulation

Prameter	Value
Number of failed satellites per plane	4
Number of kits per mother ship	8
Number of mother ships	6
Separation angle between satellites [degrees]	90

- **Phase 1:** As done for ADR1 after the launch, all the *mother ships* lie on the same circular parking orbit (table 3.2). Since ADR2 considers one *mother ship* working on 2 planes, 6 *mother ships* are needed for the mission. Five of them need to drift at the altitude of 500 km to reach their respective operational planes.
- **Phase 2:** After each *mother ship* has reached the correct RAAN, an Hohmann transfer will bring it to the altitude of the operational plane where a plane change manoeuvre is performed to adjust the inclination to 87.9°.
- **Phase 3:** Each *mother ship* cleans its first plane with a series of phasing manoeuvres.
- **Phase 4:** Each *mother ship* moves to the next operational plane by changing its inclination to 87.67° (value selected from several analysis iterations) and exploiting J_2 effect for RAAN drifting. At the end of the drift phase a second plane change manoeuvre will bring the value of inclination from 87.67° back to 87.9°, so two plane change manoeuvres need to be considered.
- **Phase 5:** Each *mother ship* cleans its second plane with a series of phasing manoeuvres.

Simulation results and discussion

There are no significant differences between ADR1 and ADR2 in the first phase, both exploiting J_2 for RAAN separation. The first *mother ship* can begin the orbit raising after the release by the launcher, while the other five are supposed to drift towards their operational planes. Again, to decrease the time of this phase, a possible solution is to decrease the release altitude or the inclination. The major drawback will be an increase of the Δv implied in the next phase. Results of the orbit raising through the Hohmann transfer and inclination change are shown in figures 3.4 and 3.5. In figure 3.19 it is possible to observe how all the *mother ships* do not respect the requirement related to the lifetime. Indeed, all of them take more than 2 years to complete the mission. The main factor affecting the total mission time is caused by **phase 4**. Each *mother ship* takes almost 2 years to drift from the first plane to the second one. This value can be related to the small inclination change implied to accelerate up the J_2 effect. Giving a larger inclination change to decrease the drifting time, a larger amount of Δv related to **phase 4** would be required, exceeding the maximum value of Δv available onboard (1 kms). A solution could be relaxing the constraint related to the maximum lifetime.

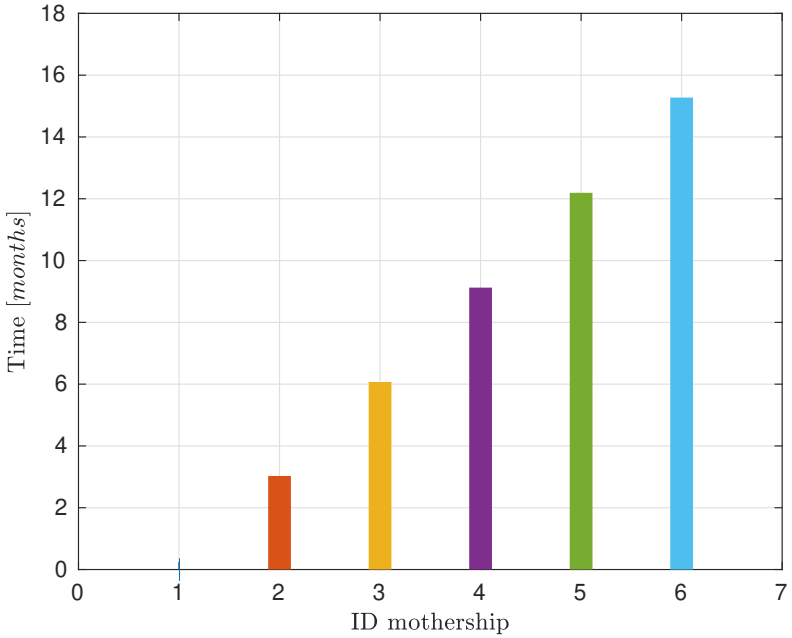


Figure 3.10: **Phase 1:** time needed for each *mother ship* to reach its operational plane

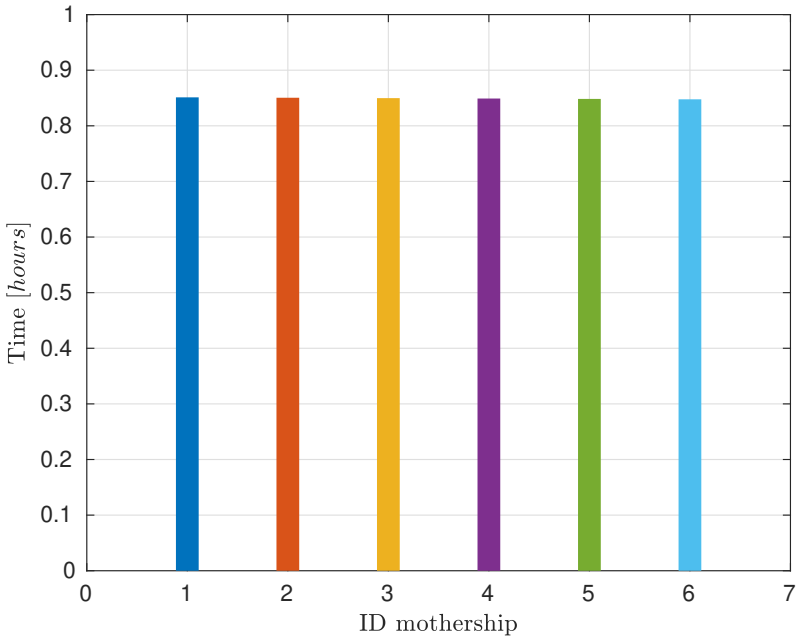


Figure 3.11: **Phase 2:** time required by the Hohmann transfer and the inclination change to complete the orbit raising

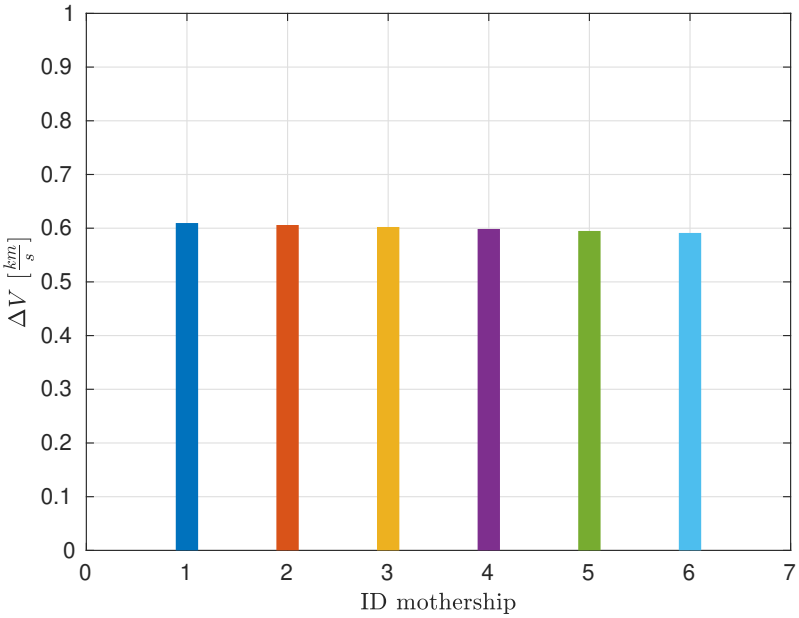


Figure 3.12: **Phase 2:** Δv required by the Hohmann transfer and the inclination change to complete the orbit raising

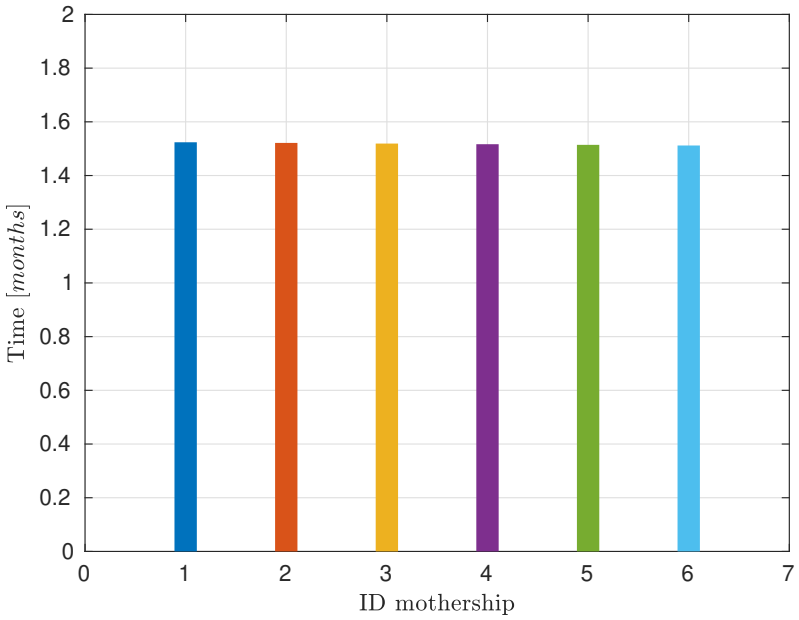


Figure 3.13: **Phase 3:** time required by each *mother ship* to reach the 4 failed satellites that lie in the first operational plane

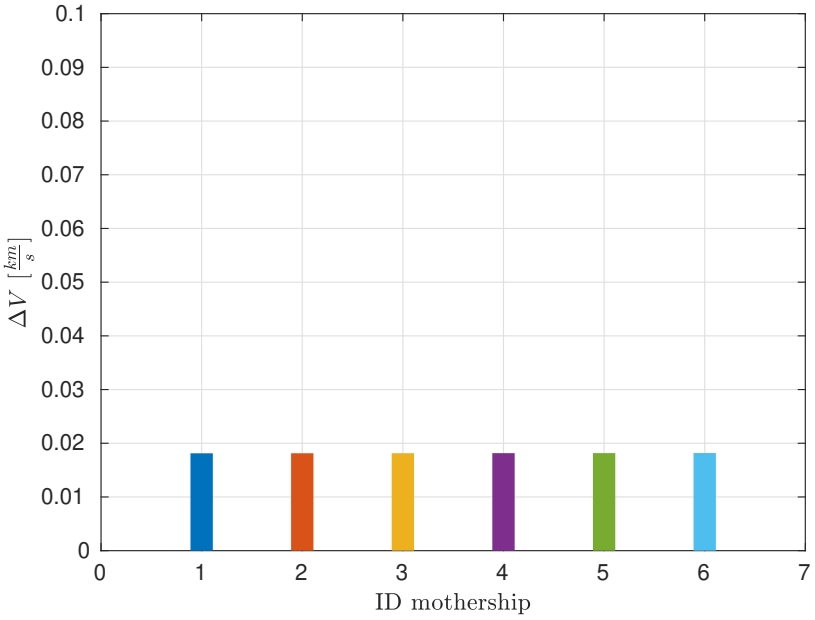


Figure 3.14: **Phase 3:** Δv required by *mother ships* to reach the 4 failed satellites that lie in the first operational plane

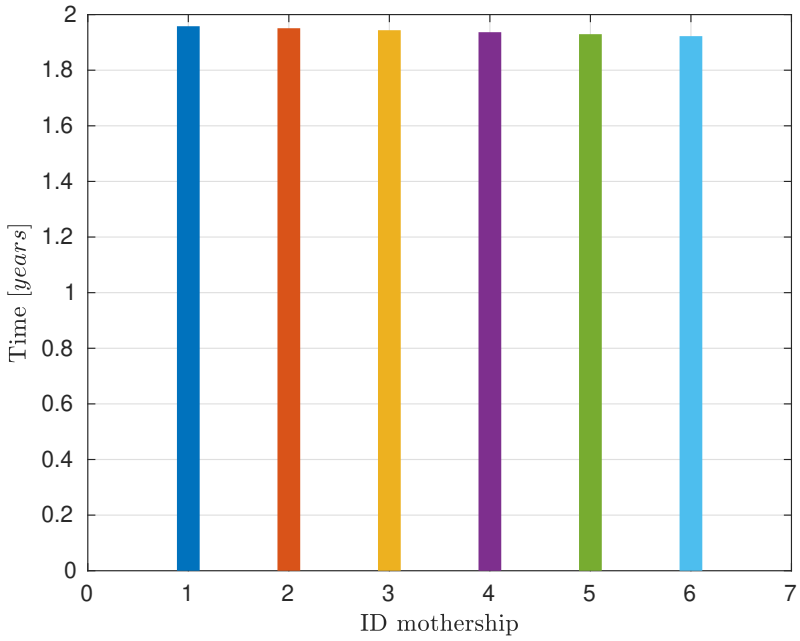


Figure 3.15: **Phase 4:** time required by the *mother ships* to reach the second plane by exploiting J_2 effect

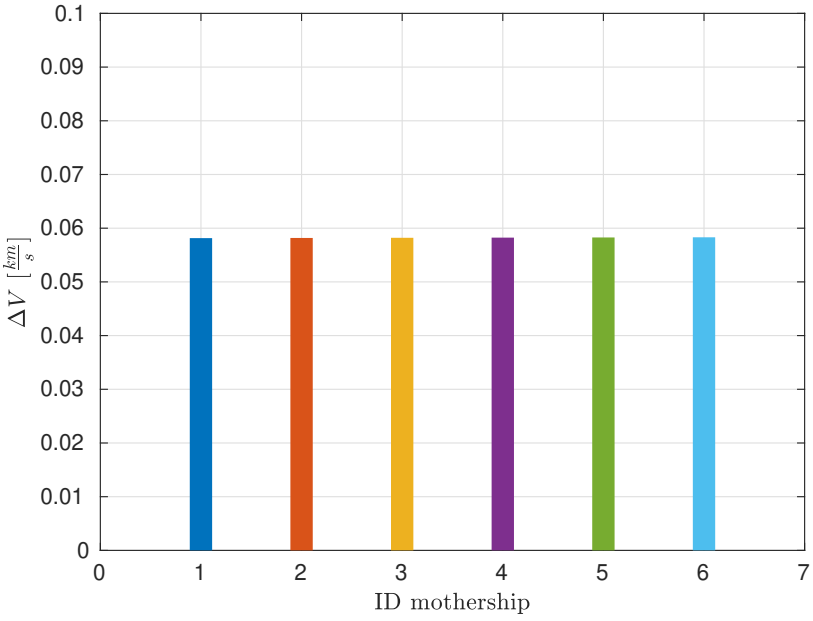


Figure 3.16: **Phase 4:** Δv required by the *mother ships* to reach the second plane by exploiting J_2 effect

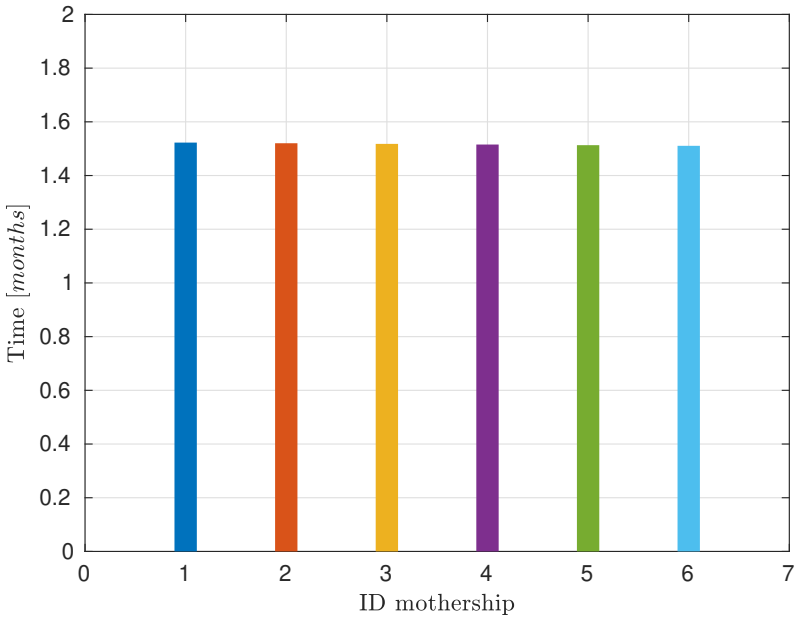


Figure 3.17: **Phase 5:** Δv required by each *mother ship* to reach the 4 failed satellites that lie in the second operational plane

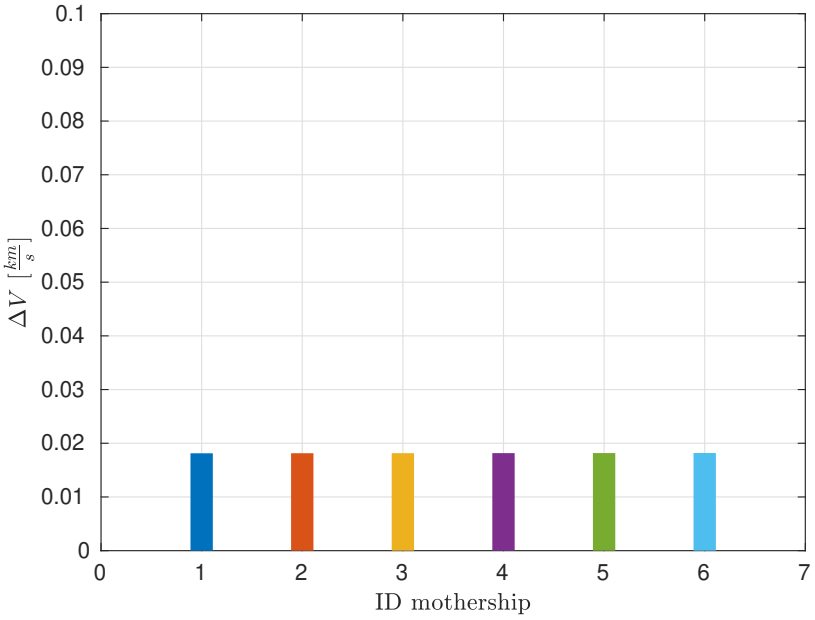


Figure 3.18: **Phase 5:** Δv required by each *mother ship* to reach the 4 failed satellites that lie in the second operational plane

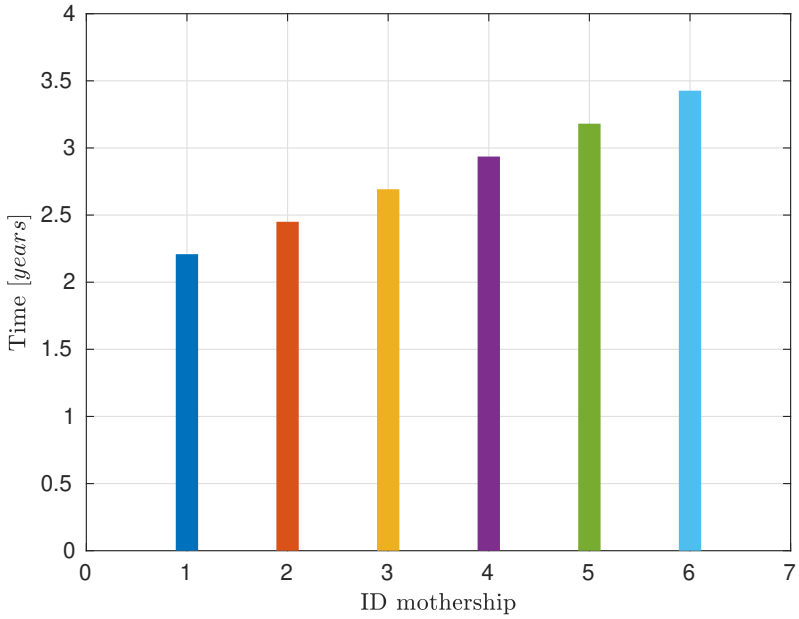


Figure 3.19: Total mission time required by each *mother ship*

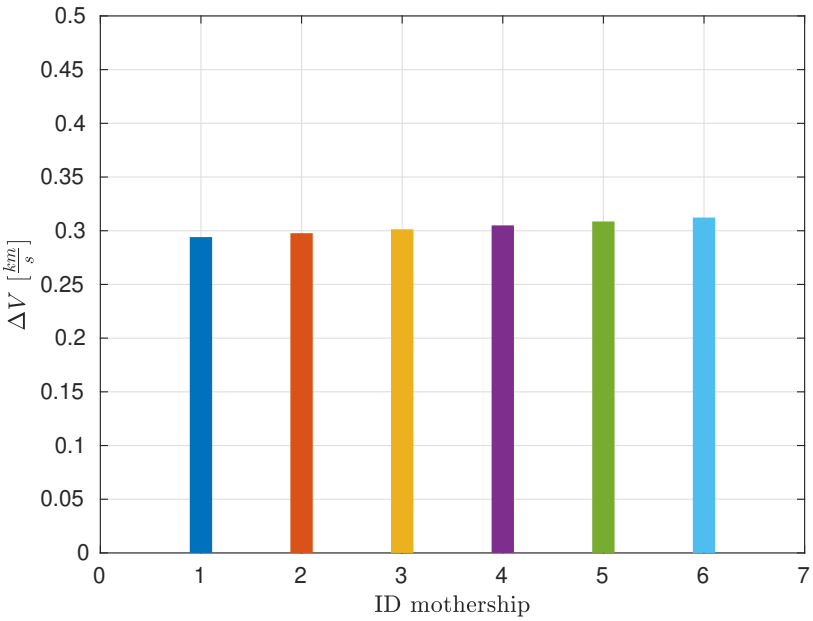


Figure 3.20: Residual Δv available to de-orbit onboard each *mother ship*

Chapter 4

Optimal design for architecture 2 based on travelling salesman approach

This chapter will design the second architecture by solving a *Travelling Salesman Problem* (TSP) within the context of graph theory. In mathematics, graph theory is a specific field of study related to combinatorics, which can be exploited to solve numerically complex optimisation problems, concerning different fields of study. In this work, it is used to solve optimisations related to the design of an efficient *Active Debris Removal* service. The modelling of the ADR problem is anticipated by an overview about graph theory and existing TSP solving procedures.

4.1 Graph theory

Graph theory is devoted to the study of graphs, mathematical structures defining the pairwise relations between objects. Graphs are composed by points called nodes or vertices, connected by lines called edges or arcs, as shown in figure 4.1.

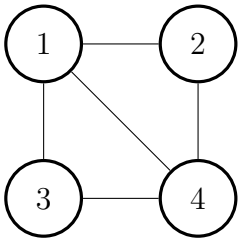


Figure 4.1: Representation of a graph characterised by four nodes and five edges

A generic graph G, composed by V vertices and E edges, is defined as $G = (V, E)$. Thus, an example of graph $G = (4, 5)$ is shown in figure 4.1. This is not the only way to define a graph; indeed it is possible to declare it as:

- $G = (V, E)$
- $G = (V(G), E(G))$
- $G = \{12, 13, 34, 24, 14\}$

the last declaration, referring to example in figure 4.1, gives more information with respect to the previous two. Indeed, it describes all the connections among the nodes providing an idea on how the graph is structured. Each number represents an edge connecting two nodes, clarifying how the points are related one to each other. Another interesting way to define a graph is:

$$G = (n, f(n)) \tag{4.1}$$

where n is the generic node of the graph and $f(n)$ is a function describing the connection rules among the nodes. It is the most compact and meaningful way to define the graph since it is expressed by the mathematical function $f(n)$. Here is an example for for this definition.

$$G = (n, 2n) \quad with\ n = 2 \tag{4.2}$$

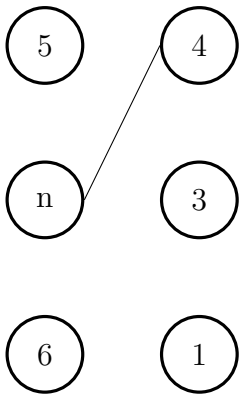
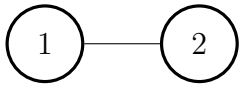


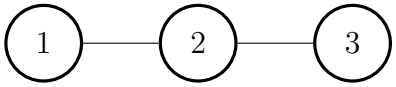
Figure 4.2: Example of the graph $G = (n, 2n)$ for $n = 2$

Nodes and vertices can have different properties inside the graph. A list of fundamental properties is listed below:

- **Adjacent nodes**: nodes connected by an arc are called *adjacent nodes*

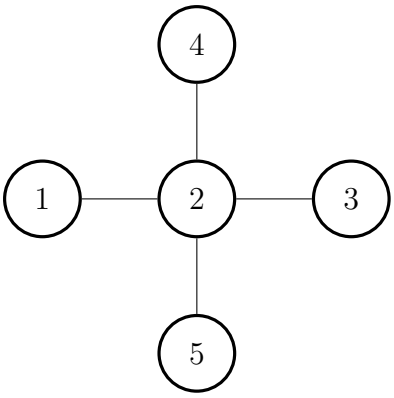


- **Adjacent arcs**: arcs with a common node are called *adjacent arcs*



  arcs connecting $1-2$ $2-3$ are adjacent arcs with the common node $\{2\}$

- **Neighbourhood**: a *neighbourhood* of $\{j\}$ includes all the adjacent nodes to a generic node $\{j\}$



$\{1, 3, 4, 5\}$ are the *neighbourhood* of $\{2\}$

The dimension of a graph is identified as the number of arcs inside of it. Calling m as the number of arcs in a graph, the range of m is:

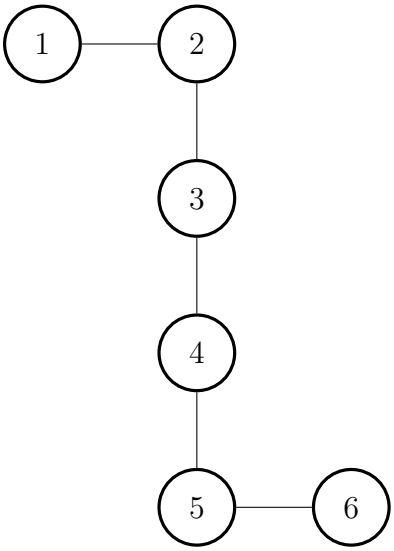
$$0 \leq m \leq \binom{n}{2} \tag{4.3}$$

where n is the number of graph nodes and the binomial coefficient is defined as:

$$\binom{n}{r} = \frac{n!}{(n-r)!\,r!} \tag{4.4}$$

A **complete graph** is a graph with the maximum possible number of arcs. Thus, each node is connected with all the remaining nodes in the graph.

- **Path**: a *path* is the sequence of edges going from a node to another. An example of the path from node $\{1\}$ to $\{6\}$ is represented below.



- **Independent paths**: two different paths are called *independent paths* if there is no intersection between them.
- **Cycle or close path**: a path having the same node as starting and arrival point is called a *cycle*.

- **Connected graph**: a graph is called *connected graph* if there exist a path between any pair of nodes.

There are two more important definitions describing graphs, useful for the future analysis of the ADR service: the *orientation* of a graph and the *adjacency matrix*. A graph is called *oriented graph* when it is composed by edges which can only be covered in one direction, while the *adjacency matrix* is a matrix composed by zeros and ones identifying the connections between nodes.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.5}$$

It needs to be symmetric due to the structure of the problem, node $\{1\}$ is connected with node $\{3\}$ and vice versa. Matrix A represents a graph like $G = \{13, 23, 34\}$. Concerning the ADR service to be designed in this chapter, since it is always possible to transfer from a satellite to another one, the *adjacency matrix* is filled with ones except for the diagonal.

4.2 The Travelling Salesman Problem

In computer science the *Travelling Salesman Problem* (TSP) is one of the most famous routing and scheduling problem. It tries to answer to the question: "Given a prescribed set of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin one?" It is one of the most important decision making-problem and it is used in many fields of study: microchip design, logistic organisation, biology an many others. The solving procedure of the TSP is used also as benchmark for optimisation algorithms, due to the high computational cost of the problem.
From the mathematical point of view the TSP can be seen as a graph where cities represent the nodes and the streets that connect them represent the edges. The objective function to be minimised is the sum of the distances of the paths that connect all the cities without any repetition. Let n be the number of cities and assume that each city is connected with all the others. The dimension of the problem is $n!$. Indeed, a tree representing all the possible choices can be built as shown in figure 4.3. The example refers to a situation where $n = 3$.

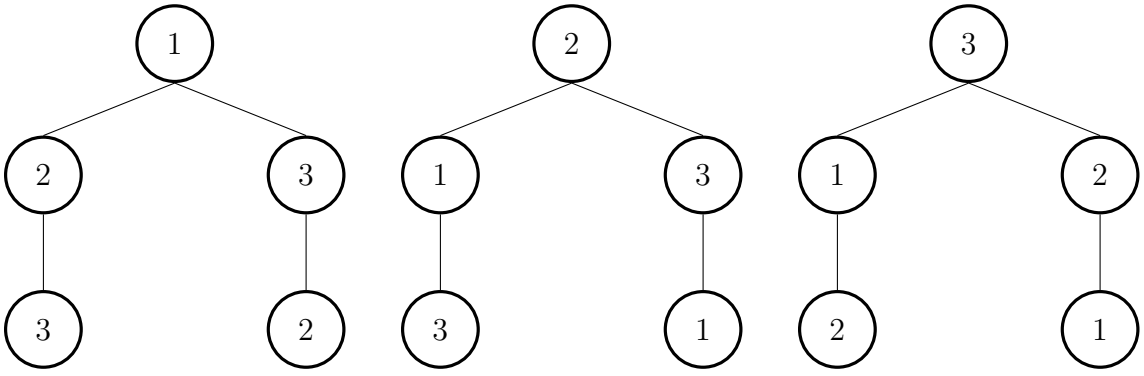


Figure 4.3: Solution tree of a TSP with $n = 3$

As indicated in figure 4.3, if any city can be connected with the others, six possible paths should be calculated to select the best one. It is straightforward to understand how exponentially the computational cost of such a problem can increase when n becomes larger. Due to this fact, it is not feasible to compute all the possible combinations, even exploiting a powerful computer. Other solution methods should be used to solve the problem.

4.3 Mission design

The second architecture that has been introduced in chapter 2 can be modelled as an Asymmetric TSP (ATSP). The mission steps 2,3 and 4 described in section 2.1.2 refer to a similar situation with respect to the TSP. The first one concerning the initial orbit raising is not taken into consideration in this analysis. The *chaser* is in charge of reaching all the dead spacecraft and moving them to the disposal orbit. In this case, the *chaser* is the travelling salesman and the dead satellites are the cities to be visited. Then the question is: "which is the best order to reach all the satellites that minimise the total mission time?". Since the velocity change to move from a failed spacecraft to the next is fixed by the manoeuvres described in sections 2.3.2 and 2.3.4, it is not an objective of the optimisation. Indeed, once the characteristics of the disposal orbit are prescribed, transfers are not tunable and are identified by a single value of Δv. It is clear that, since satellites do not lie on a two dimensional plane and the path going from one to the next is not a straight line, calculating the costs is more complicated with respect to the classical TSP. Moreover, exploiting the transfers described in sections 2.3.2 and 2.3.4 , the problem becomes asymmetric. Indeed, the cost to move from the generic satellite A to B is not equal to the transfer from B to A. In this chapter three trial constellations are taken as case studies: *OneWeb*, *Starlink* and *Globalstar*. An optimal solution of the ADR service is computed by exploiting three different methods that have been introduced in section 1.2.2:

- *Brute Force Algorithm*
- *Branch and Cut Algorithm*
- *Nearest Neighbour Algorithm*

The feasibility and the computational time of all the three methods will be analised to find the one with the best performances in terms of the CPU time. All the simulations are run on a workstation with the following characteristics:

- CPU: Intel®Core i7-9750H CPU @ 2.60GHz × 12
- RAM: 16GB of physical memory + 16GB of virtual memory (swap)
- OS: Ubuntu 20.04 LTS

The program used to run the simulation is MATLAB, version R2020a.

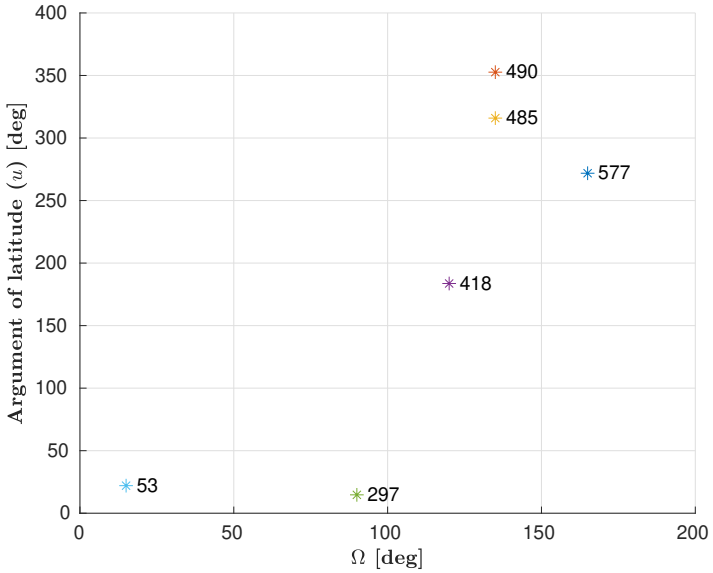


Figure 4.4: u *vs* Ω representation for the OneWeb constellation. Satellites are sorted from the first to the sixth in a descending order. Number 577 represents the first and number 53 the last one.

4.3.1 Problem statement

Firstly, the problem domain has to be defined. Considering a circular-orbit constellation, at given altitude and inclination, each satellite can be identified using only two parameters, the right ascension of the ascending node Ω and the argument of latitude u. An example representation of six randomly selected dead satellites inside the *OneWeb* constellation is shown in figure 4.4. Secondly, the costs of the edges have to be calculated. To compute this values, all the possible transfers among the satellites are simulated and the output data are stored inside a $n \times n$ matrix. For example the cell corresponding to the first row and second column stores the cost of the transfer starting from the first satellite to the second. Obviously, the matrix is non-symmetric and has an empty diagonal. Equation 4.6 is an example of the cost matrix referring to the case shown in figure 4.4. The costs concerning the transfer times are expressed in years.

$$C = \begin{bmatrix} 0 & 1.63928 & 1.6411 & 2.45 & 4.09 & 8.18 \\ 18.0 & 0 & 0.0019 & 0.82 & 2.45 & 6.54 \\ 18.0 & 0.00258 & 0 & 0.82 & 2.45 & 6.54 \\ 17.1 & 18.8253 & 18.827 & 0 & 1.63 & 5.72 \\ 15.5 & 17.1877 & 17.187 & 18.0 & 0 & 4.09 \\ 11.4 & 13.0955 & 13.095 & 13.9 & 15.5 & 0 \end{bmatrix} \tag{4.6}$$

4.3.2 Simulation code

The simulation code is written in MATLAB. The code is built in a modular way so that it can be simply modified and improved, without the need to re-write from scratch. In this way it is possible to run the code for any constellation, changing only the input file. The following assumptions are considered for the simulations:

- The input constellation is composed by circular orbits
- The input constellation has a common inclination for all orbital planes

In the code, the variables are stored in the structure format for the convenience of being passed to the sub-functions. To compare the performances of the three solving methods used, two different scenarios are considered.

- **Scenario 1**: failures are generated randomly over all orbital planes.
- **Scenario 2**: failures are generated randomly in only three orbital planes with at least two dead spacecraft per plane. In this scenario, the failure density within a single orbital plane is comparatively high. The aim of considering this scenario is to investigate the performances of the three algorithms in case of a higher failure density.

An additional Monte Carlo simulation is conducted to compare the performances and the quality of the three solution methods.

Input module

The three input files used for the two scenarios are listed below. The input module of the code consists of an *input* function collecting astronomical constants, orbital parameters of the constellations and design data. The design data refer to the characteristics of the disposal orbit introduced in section 2.3.4. Since the spacecraft is supposed to re-enter exploiting drag perturbation, perigee and apogee radii are calculated by *PlanODyn*, an in-house semi-analytical orbit propagator [2]. They are different for each constellation (table 4.1). The disposal orbit is designed such that

Table 4.1: Disposal orbit parameters

Constellation	Perigee altitude [km]	Apogee altitude [km]
OneWeb	351	1100
Globalstar	338	1300
Starlink	368	450

the dead satellites can re-enter within 5 years under the drag.

```
function [INPUT] = input_mpOW()
%INPUT_MPOW Input data function

ac = astroConst();
INPUT.Re = ac.R_Earth;
INPUT.mu = ac.mu_Earth;
INPUT.J2 = ac.J2_Earth;
[INPUT.kep,INPUT.ID_mat,~] = kepOW();
INPUT.disposal = [INPUT.Re+351;INPUT.Re+1100;INPUT.Re+380];
end
```

```
function [INPUT] = input_mpGS()
%INPUT_MPGS Input data function

ac = astroConst();
INPUT.Re = ac.R_Earth;
INPUT.mu = ac.mu_Earth;
INPUT.J2 = ac.J2_Earth;
[INPUT.kep,INPUT.ID_mat,~] = kepGS();
INPUT.disposal = [INPUT.Re+338;INPUT.Re+1300;INPUT.Re+380];
end
```

```
function [INPUT] = input_mpSL()
%INPUT_MPSL INput data function

ac = astroConst();
INPUT.Re = ac.R_Earth;
INPUT.mu = ac.mu_Earth;
INPUT.J2 = ac.J2_Earth;
[INPUT.kep,INPUT.ID_mat,~] = kepSL();
INPUT.disposal = [INPUT.Re+368;INPUT.Re+450;INPUT.Re+380];
end
```

The only difference between the input functions for the two scenarios is the number of constellations planes considered. In the first case all the constellation planes are considered, while in the second one only the first three planes are taken into consideration.

```
function [INPUT] = input_3pOW()
%INPUT_3POW Input data function

ac = astroConst();
INPUT.Re = ac.R_Earth;
INPUT.mu = ac.mu_Earth;
INPUT.J2 = ac.J2_Earth;
[INPUT.kep,INPUT.ID_mat,~] = kepOW();
INPUT.ID_mat = INPUT.ID_mat(1:3,:);
INPUT.kep = INPUT.kep(1:INPUT.ID_mat(end,end),:);
INPUT.disposal = [INPUT.Re+351;INPUT.Re+1100;INPUT.Re+380];
end
```

```
function [INPUT] = input_3pGS()
%INPUT_3PGS Input data function

ac = astroConst();
INPUT.Re = ac.R_Earth;
INPUT.mu = ac.mu_Earth;
INPUT.J2 = ac.J2_Earth;
[INPUT.kep,INPUT.ID_mat,~] = kepGS();
INPUT.ID_mat = INPUT.ID_mat(1:3,:);
INPUT.kep = INPUT.kep(1:INPUT.ID_mat(end,end),:);
INPUT.disposal = [INPUT.Re+351;INPUT.Re+1100;INPUT.Re+380];
end
```

```
function [INPUT] = input_3pSL()
%INPUT_3PSL Input data function

ac = astroConst();
INPUT.Re = ac.R_Earth;
INPUT.mu = ac.mu_Earth;
INPUT.J2 = ac.J2_Earth;
[INPUT.kep,INPUT.ID_mat,~] = kepSL();
INPUT.ID_mat = INPUT.ID_mat(1:3,:);
INPUT.kep = INPUT.kep(1:INPUT.ID_mat(end,end),:);
INPUT.disposal = [INPUT.Re+300;INPUT.Re+450;INPUT.Re+300];
end
```

Modelling module

This module models the problem. The inputs of the modelling function are the *INPUT* structure, built in the input module, and the number of dead spacecraft considered. Failures are generated randomly according to the scenario taken into consideration. In the first scenario, the function randomly selects a uniform distribution of failures over all orbital planes. In the second scenario, the function is constrained to generate at least two failed satellites per plane. Moreover, the modelling function is in charge of evaluating all the possible transfers creating the cost matrix. Two possible transfers are possible depending on the position of the next satellite:

1. The next satellite lies in the same orbital plane of the *chaser*. In this case, a bitangent manoeuvre is performed to move the current satellite to the disposal orbit (section 2.3.4) and a subsequent orbit raising moves the *chaser* towards the next satellite (section 2.3.2).

2. The next satellite lies in a different orbital plane of the *chaser*. In this case, after the disposal manoeuvre is performed, the *chaser* is let to drift towards the target orbital plane by exploiting J_2 perturbation (section 2.3.3). No additional Δv is is provided to the *chaser* during this phase. When the correct RAAN is reached by the *chaser*, an orbit raising manoeuvre moves it towards the next target (section 2.3.2).

It is clear that the Δv is fixed and equal for each transfer, and they differ only in terms of time of flight . Thus, the only variable to be minimised is the transfer time, which represents the cost of each transfer. The maximum number of failed spacecraft considered is 12, due to the computational costs and time of the simulation. Using more powerful technologies, like a cluster of multiple CPUs, it is possible to run simulations for large-scale problems. The codes for the two scenarios are presented in the next pages.

```
function [MODEL] = model_mp(INPUT, n_fail)

MODEL.ID_mat = INPUT.ID_mat';
[r,c] = size(MODEL.ID_mat);
MODEL.ID_mat = reshape(MODEL.ID_mat,1,r*c);
MODEL.fails = randperm(r*c, n_fail);
MODEL.fails = sort(MODEL.fails);
MODEL.fails = fliplr(MODEL.fails);
MODEL.kep = INPUT.kep(MODEL.fails,:);
MODEL.n_fail = n_fail;

kep = MODEL.kep;
dt = zeros(n_fail, n_fail);

for k = 1:n_fail
    kep_int = kep(k,:);
    for kk = 1:n_fail
        if k == kk
            dt(k,kk) = nan;
        elseif k ~= kk
            kep_tgt = kep(kk,:);
            [~,dt(k,kk),~,~] = adrTransfer(kep_int,kep_tgt,
               INPUT);
        end
    end
end

cmat = dt/year2sec;
MODEL.cmat = cmat;

end  % function
```

```
function [MODEL] = model_3p(INPUT, n_fail)

MODEL.ID_mat = INPUT.ID_mat';
[r,c] = size(MODEL.ID_mat);
MODEL.ID_mat = reshape(MODEL.ID_mat,1,r*c);
nMin = 2;
flag = 0;
while flag == 0
    fails = randperm(r*c, n_fail);
    for k = length(fails):-1:1
        [fplane(k),~] = find(INPUT.ID_mat == fails(k));
    end
    A = accumarray(fplane(:),1);
    flag = 1;
    for k = 1:length(A)
        if A(k) < nMin
            flag = 0;
        end
    end
end
MODEL.fails = fails;
MODEL.fails = sort(MODEL.fails);
MODEL.fails = fliplr(MODEL.fails);
MODEL.kep = INPUT.kep(MODEL.fails,:);
MODEL.n_fail = n_fail;

kep = MODEL.kep;
dt = zeros(n_fail, n_fail);

for k = 1:n_fail
    kep_int = kep(k,:);
    for kk = 1:n_fail
        if k == kk
            dt(k,kk) = nan;
        elseif k ~= kk
            kep_tgt = kep(kk,:);
            [~,dt(k,kk),~,~] = adrTransfer(kep_int,kep_tgt,
                INPUT);
        end
    end
end

cmat = dt/year2sec;
MODEL.cmat = cmat;

end  % function
```

Solver module

This is the most important part of the code. The problem is solved exploiting the three algorithms selected. Here is a detailed description of all the methods adopted.

Brute Force Algorithm This solver tries all the possible combinations to get the best one. Within the scope of the current ATSP, all the possibilities are represented by the permutations of n elements, where n is the number of failures to be reached. To compute the permutations, a built-in MATLAB function called *perms* is used. The function *perms* returns a matrix in which each row represents a permutation of the elements of the input array. A *for* cycle is exploited to evaluate the cost of each permutation. During each iteration the cycle selects a row of the matrix and compute the cost of the sequence. This method can always guarantee to find the best solution but it is not suitable for large-scale problems. Indeed, the major drawback of this approach, in addition to the high computational time, is the amount of memory required. Data inside the matrix P, containing all the possible permutations, are stored as variables of type *double*. A variable of type *double* requires 8 bytes to be stored. With this information it is not difficult to calculate the total memory a matrix is supposed to occupy. For instance the P matrix related to a problem with $n = 3$ is:

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \tag{4.7}$$

P has $n!$ rows and n columns. Thus, it consists of $n! \times n$ elements of type *double*, each requiring 8 bytes. The total memory required is:

$$RAM = n! \times n \times 8 \text{ bytes} \tag{4.8}$$

From equation 4.8, when $n = 3$, $RAM = 144$ bytes but it is straightforward to understand how RAM can increase exponentially. Indeed, when $n = 12$ the total memory required is 42.8 Gigabytes. Looking at the code, the algorithm is composed by two nested simple *for* cycles. The external one selects the sequence to test and the internal one calculates the cost of the branch. Computational time is very low for small values of n but it increases rapidly as n increases. Indeed, only 0.0007 seconds are required to solve a graph with 7 vertices, while for an instance with 10 vertices it takes 0.7 seconds. Increasing the dimension of three units the time is increased on the order of 10^3. Thus, solving an ATSP through the brute force method with a larger n will be unfeasible. Despite the limitations on the memory can be avoided generating the branches one by one, instead of storing all of them in a matrix, the bottleneck related to the computational time cannot be neglected.

The code is presented here.

```
function [SOLUTION] = solver_bf(MODEL)

kep = MODEL.kep;
[r,~] = size(kep);
cmat = MODEL.cmat;
P = perms(1:r);
[n_comb,c] = size(P);
SOLUTION = zeros(n_comb,1);

for k = 1:n_comb
    p = P(k,:);
    cost = 0;
    for kk = 1:(c-1)
        cost_temp = cmat(p(kk),p(kk+1));
        cost = cost + cost_temp;
    end
    SOLUTION(k) = cost;
end
SOLUTION = [P,SOLUTION];
end  % function
```

Nearest Neighbour Algorithm Due to its simplicity, the nearest neighbour algorithm is one of the prime algorithms used to solve the TSP. Instead of selecting randomly the initial point, the NNA is executed as many times as the number of problem vertices. In this way, each city is selected one time as the starting point of the algorithm. The following steps summarise the attached code:

1. Choosing a starting vertex.
2. Through the cost matrix the nearest vertex is selected as the next one and the corresponding row is eliminated, marking the city as visited.
3. Repeating Step 2 until all the vertices are visited.
4. A new starting vertex is chosen and repeating Step 1 to 3.
5. Once all the NNA sequences have been calculated the one with the lowest cost is selected as the optimum.

The code is presented here.

```
function [SOLUTION] = solver_nna(MODEL)

cmat = MODEL.cmat;
n_fail = MODEL.n_fail;
comb = zeros(n_fail,n_fail);
val = zeros(n_fail,1);

for k = 1:n_fail
    ind = k;
    mat = cmat;
    n_t = 0;
    comb(k,1) = ind;
    while n_t < (n_fail - 1)
        mat(:,ind) = nan;
        [~,ind2] = min(mat(ind,:));
        temp = mat(ind,ind2);
        n_t = n_t + 1;
        comb(k,n_t + 1) = ind2;
        val(k) = val(k) + temp;
        ind = ind2;
    end
end

SOLUTION= [comb,val];

end  % function
```

Branch and Cut algorithm This algorithm is derived from the *branch and bound* theory, and it is the specific algorithm used to solve this problem is based on a *pruning* action. Pruning is a common approach used in branch and bound algorithm to reduce the size of the graph analysed. A subset of branches are evaluated a priori as non optimal and are discarded by the algorithm to reduce the computational costs. Based on J_2 perturbation theory and manoeuvres design described in chapter 2, it has been observed that optimal solutions are always found starting from spacecraft lying on the last orbital planes. Indeed, when the *chaser* is let to drift after the disposal of one satellite, it tends to move naturally towards the next plane characterised by a lower RAAN. Thus, it is possible to drive the algorithm to discard the sequences which include transfers towards planes with larger values of RAAN. Branches are generated one by one and then checked. If the branch does not belong to the "optimal solutions family", it is discarded a priori without being evaluated. Within the set of branches evaluated the one with the lowest cost is marked as the best solution. Branches generation is a crucial part of the algorithm, due to the fact that permutations are not created using the built-in function *perms*, avoiding the bottleneck due to RAM demand. The method used to generate the sorted permutations is based on the work of the mathematician Derrick Norman Lehmer. The *Lehmer code* is able to associate an integer number to a specific permutation of an elements set [8]. For instance, it is known that the total number of permutations given a set of n elements is $n!$. If $n = 4$ the question is: It is possible to associate the rank of a permutation with the corresponding one? It can be done exploiting the *factoradic* method. The integer number of the rank can be expressed as a polynomial in which the base is composed by the factorial numbers going from 0 to $n - 1$. Referring to the example in table 4.2, rank 16 can be expressed as:

$$16 = 2 \times 3! + 2 \times 2! + 0 \times 1! + 0 \times 0! \tag{4.9}$$

The coefficients of the base elements compose the so called *factoradic code*. Converting all the ranks into *factoradic codes* table 4.3 is obtained. Once the *factoradic code* has been generated, with a few steps it is possible to convert it to the corresponding permutation. For instance taking rank 16 the associated factoradic code is 2200. All the values in the factoradic array indicate how many lower numbers there are to the right of that specific position, with respect to the number stored in that position. Number 2 in the first position indicates that the first number of the sequence is greater than two number at its right. The same for position two. The permutation is then $[3, 4, 1, 2]$. Indeed, the number in position one $\{3\}$ is greater than two numbers to its right $\{1, 2\}$, and the same for number $\{4\}$ in position two. Exploiting this method, all the paths of the graph can be generated, tested and evaluated one by one, without the need to store them. To speed up the code it can be run in parallel pool. Parallel computation is faster with respect to the sequential one when the cycles involved requires a lot of time for the iterations. After the branch generation a pruning algorithm decides if the branch should be evaluated or not. To understand better the following example concerning 10 dead satellites inside the *OneWeb* constellation is considered. Figure 4.5 shows the problem domain. Numbers identify the identity of the failed satellites inside the constellation. Ten satellites randomly selected inside three orbital planes are considered. Since the chaser can only drift backwards (see section 2.3.3), the most convenient choice is to clean each plane before moving to next one, starting from the last plane that

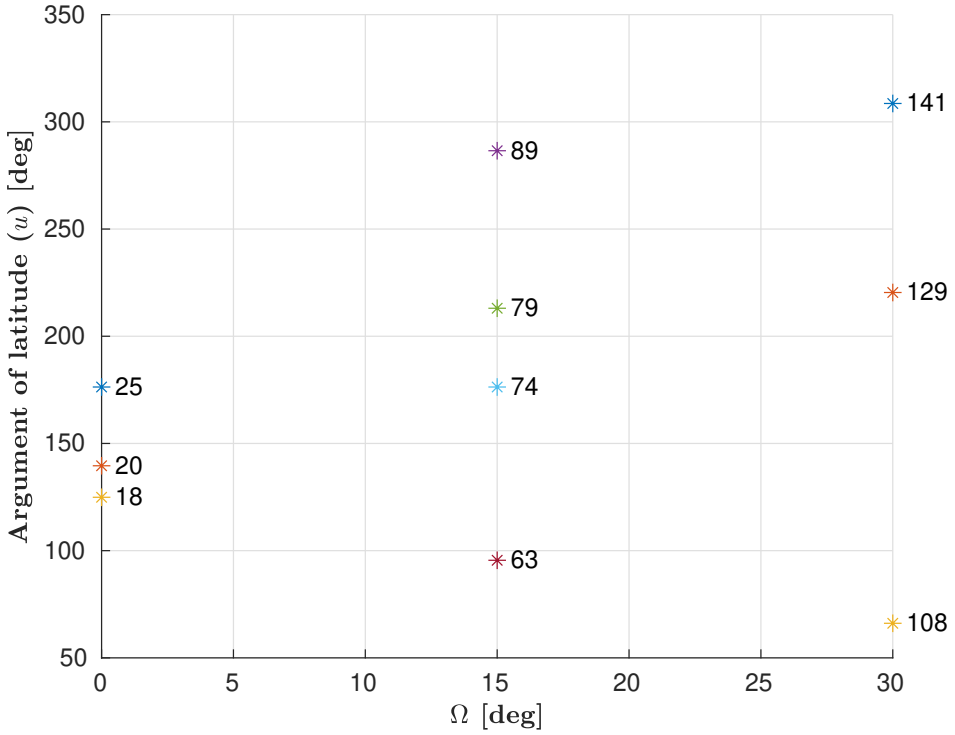


Figure 4.5: Representationof 10 dead satellites inside the *OneWeb* constellation. The satellites are sorted from the first to the last in a descending order. Satellite 141 is considered to be the failed satellite number 1 and satellite 18 the failed satellite number 10.

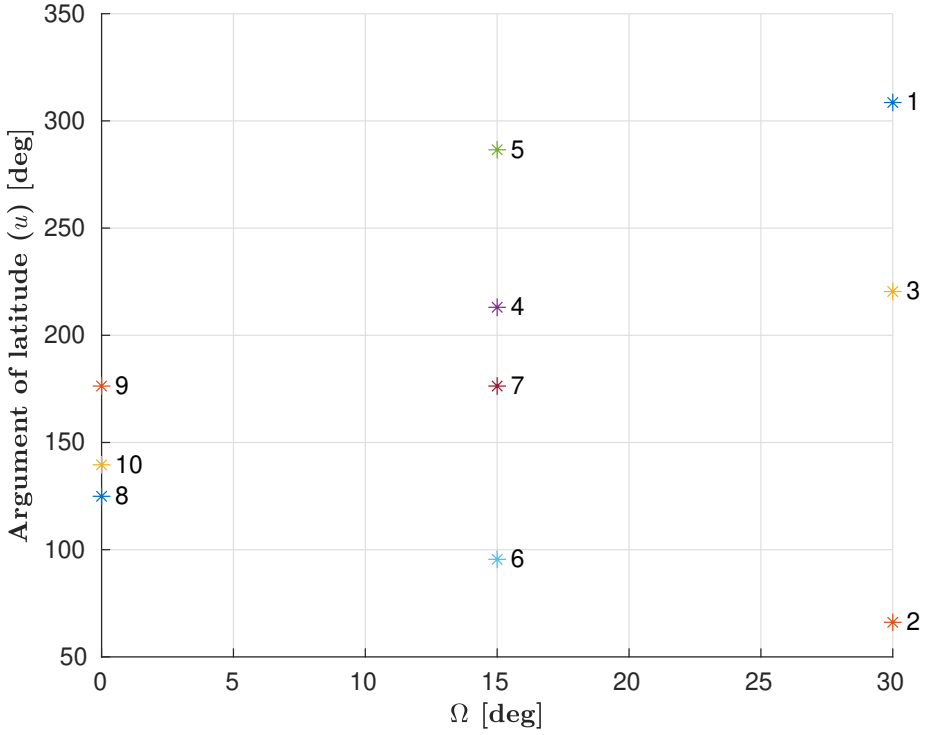


Figure 4.6: Solution sequence of the $n = 10$ example

Table 4.2: Permutations related to a 4 elements set

Rank	Permutation
0	$[1, 2, 3, 4]$
1	$[1, 2, 4, 3]$
2	$[1, 3, 2, 4]$
3	$[1, 3, 4, 2]$
4	$[1, 4, 2, 3]$
5	$[1, 4, 3, 2]$
6	$[2, 1, 3, 4]$
7	$[2, 1, 4, 3]$
8	$[2, 3, 1, 4]$
9	$[2, 3, 4, 1]$
10	$[2, 4, 1, 3]$
11	$[2, 4, 3, 1]$
12	$[3, 1, 2, 4]$
13	$[3, 1, 4, 2]$
14	$[3, 2, 1, 4]$
15	$[3, 2, 4, 1]$
16	$[3, 4, 1, 2]$
17	$[3, 4, 2, 1]$
18	$[4, 1, 2, 3]$
19	$[4, 1, 3, 2]$
20	$[4, 2, 1, 3]$
21	$[4, 2, 3, 1]$
22	$[4, 3, 1, 2]$
23	$[4, 3, 2, 1]$

has the largest RAAN. The idea is to reduce the $n!$ combinations, to a much lower value, basing on heuristic observations. The first group to be reached by the chaser will be $\{108, 129, 141\}$, the second one $\{63, 74, 79, 89\}$ and the last one $\{18, 20, 25\}$. Pruning all the combinations not respecting this rule the number of sequences to be evaluated becomes:

$$N = 3 \times 4 \times 3 = 36 \tag{4.10}$$

which is much more smaller than 10!. The solution of the example in figure 4.5 is shown in figure 4.6. The algorithm is summarised in the following steps:

1. A storing matrix is generated with the aid of the Lehmer code to save the evaluated branches. The dimensions can be defined according to the amount of memory available. While the number of column is fixed by the number of vertices of the problem, the maximum number of rows can be set to not exceed a certain amount of RAM.

2. Generating the first branch.

3. Passing the branch to the pruning routine to check if it shall be evaluated or not.

Table 4.3: Factoradic codes related to a 4 elements set

Rank	Factoradic	Permutation
0	0000	[1, 2, 3, 4]
1	0010	[1, 2, 4, 3]
2	0100	[1, 3, 2, 4]
3	0110	[1, 3, 4, 2]
4	0200	[1, 4, 2, 3]
5	0210	[1, 4, 3, 2]
6	1000	[2, 1, 3, 4]
7	1010	[2, 1, 4, 3]
8	1100	[2, 3, 1, 4]
9	1110	[2, 3, 4, 1]
10	1200	[2, 4, 1, 3]
11	1210	[2, 4, 3, 1]
12	2000	[3, 1, 2, 4]
13	2010	[3, 1, 4, 2]
14	2100	[3, 2, 1, 4]
15	2110	[3, 2, 4, 1]
16	2200	[3, 4, 1, 2]
17	2210	[3, 4, 2, 1]
18	3000	[4, 1, 2, 3]
19	3010	[4, 1, 3, 2]
20	3100	[4, 2, 1, 3]
21	3110	[4, 2, 3, 1]
22	3200	[4, 3, 1, 2]
23	3210	[4, 3, 2, 1]

4. If the branch is not pruned it is evaluated and stored, otherwise switch to the next branch.

5. Repeating Step 2 to 4 until all the branches are pruned or evaluated. Whenever the matrix storing the evaluated branches is full, the best solution is stored deleting all the others. The storing matrix is cleared to to collect other possible branches.

6. At the end of the algorithm the best sequence is selected as the optimum.

The code is presented here.

```
function [ SOLUTION ] = solver_bc_pp( INPUT,MODEL )

% Getting data
IDmat = INPUT.ID_mat;
fails = MODEL.fails;
cmat = MODEL.cmat;
for k = length(fails):-1:1
    [fplane(k),~] = find(IDmat == fails(k));
end
n = length(fails);

% Analyze the orbital planes
np = 1;
nplanes = length(unique(fplane));
scmat = zeros(nplanes,n);
scmat(1,1) = 1;
scPerPlane = ones(nplanes,1);
for k = 2:n
    if fplane(k) == fplane(k-1)
        scPerPlane(np) = scPerPlane(np) + 1;
        scmat(np,scPerPlane(np)) = k;
    elseif fplane(k) ~= fplane(k-1)
        np = np + 1;
        scmat(np,scPerPlane(np)) = k;
    end
end

RAM = 3;
nC = n+1;
nR = floor(RAM*1024^3/nC/8);
fixnR = nR;
nSol = factorial(n-1)*scPerPlane(1);

totRAM = nSol*nC*8/(1024^3);
nSolver = ceil(totRAM/RAM);

counter = 1;
SOLUTION = zeros(nSolver,nC);

while counter <= nSolver
    solMAT = zeros(nR,nC);
    if counter == nSolver
        if mod(nSol,nR) == 0
        else
            nR = mod(nSol,nR);
        end
    end
```

```
48      parfor k = 1:nR
49          % Branch generation
50          numFact = (counter-1)*fixnR+(k-1);
51          [branch] = factperms(numFact,n);
52          % Pruning function
53          check = 0;
54          for kk = 1:length(branch)
55              p = 1;
56              while kk > sum(scPerPlane(1:p))
57                  p = p + 1;
58              end
59              if sum(ismember(scmat(p,:),branch(kk))) > 0
60                  check = 1;
61              else
62                  check = 0;
63                  break;
64              end
65          end
66
67          if check == 1
68              % Test the branch
69              cost = 0;
70              for b = 1:(length(branch)-1)
71                  cost_temp = cmat(branch(b),branch(b+1));
72                  cost = cost + cost_temp;
73              end
74              solMAT(k,:) = [branch,cost];
75          elseif check == 0
76          end
77
78      end
79      [r,~] = find(solMAT(:,1) == 0);
80      solMAT(r,:) = [];
81      clear r;
82      solMAT = sortrows(solMAT,nC);
83      [rMat,~] = size(solMAT);
84      if rMat ~= 0
85          SOLUTION(counter,:) = solMAT(1,:);
86      end
87      clear solMAT;
88      counter = counter + 1;
89  end
90  [r,~] = find(SOLUTION == 0);
91  SOLUTION(r,:) = [];
92  SOLUTION = sortrows(SOLUTION,nC);
93  SOLUTION = SOLUTION(1,:);
94
95  end  % solver_bb
```

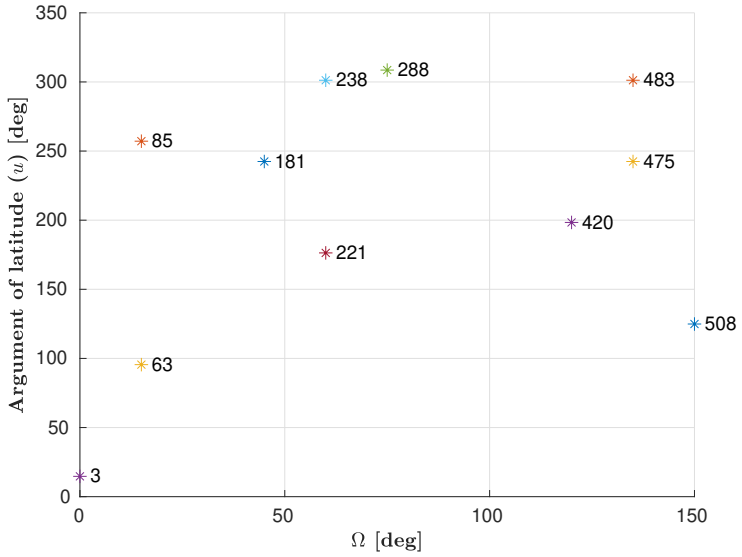


Figure 4.7: Distribution of the failures for the *OneWeb* constellation, for scenario 1

4.3.3 Simulations results and discussion

In this section the simulation results are presented and discussed. The aim of the simulations is to understand which is the best solving method in terms of quality of the solution and computational time. Two types of scenarios have been considered as discussed in section 4.3.2. In addition, a Monte Carlo simulation has been conducted to check the robustness of the algorithms. Brute force method is used only when the number of vertices of the problem was less than 12, due to memory limitations.

Scenario 1 - Failures randomly distributed over all orbital planes

OneWeb The distribution of the failures for the *OneWeb* constellation is presented in figure 4.7. The satellites are identified by an ID number inside the constellation, represented attached to the asterisks in figure 4.7. Simulation data and results are listed in table 4.4.

Table 4.4: Simulation data and results for *OneWeb* constellation for scenario 1

Parameter	Value
Constellation	OneWeb
Number of failed satellites	11
Brute force CPU time [seconds]	9.35
Branch and cut CPU time [seconds]	37.20
Nearest Neighbour CPU time [seconds]	0.000287
Brute force solution [years]	8.1999
Branch and cut solution [years]	8.1999
Nearest Neigbour solution [years]	8.2001

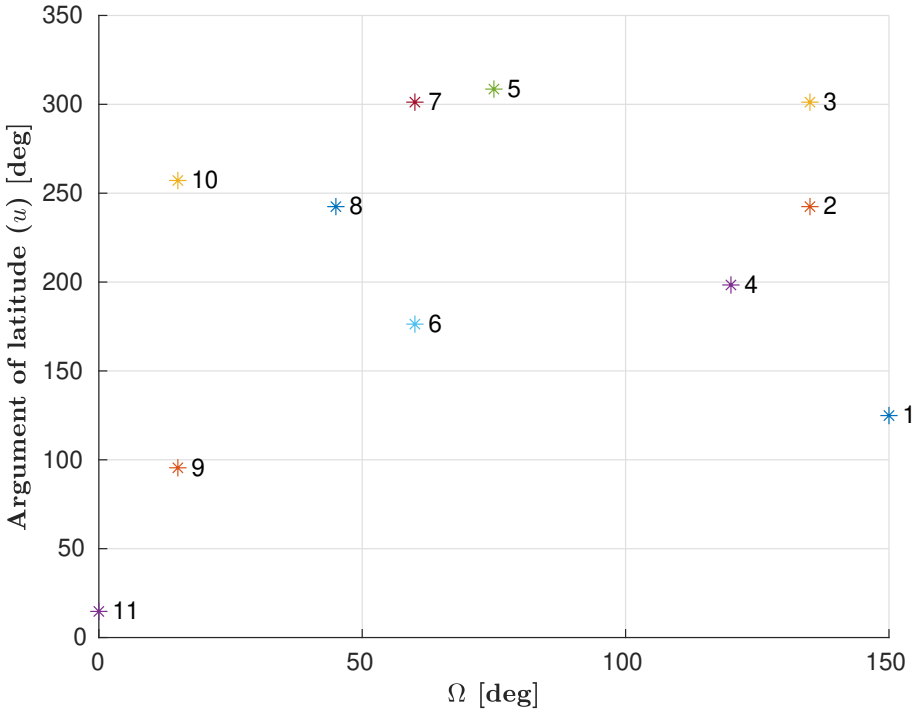


Figure 4.8: Brute force solution for the *OneWeb* constellation, for scenario 1

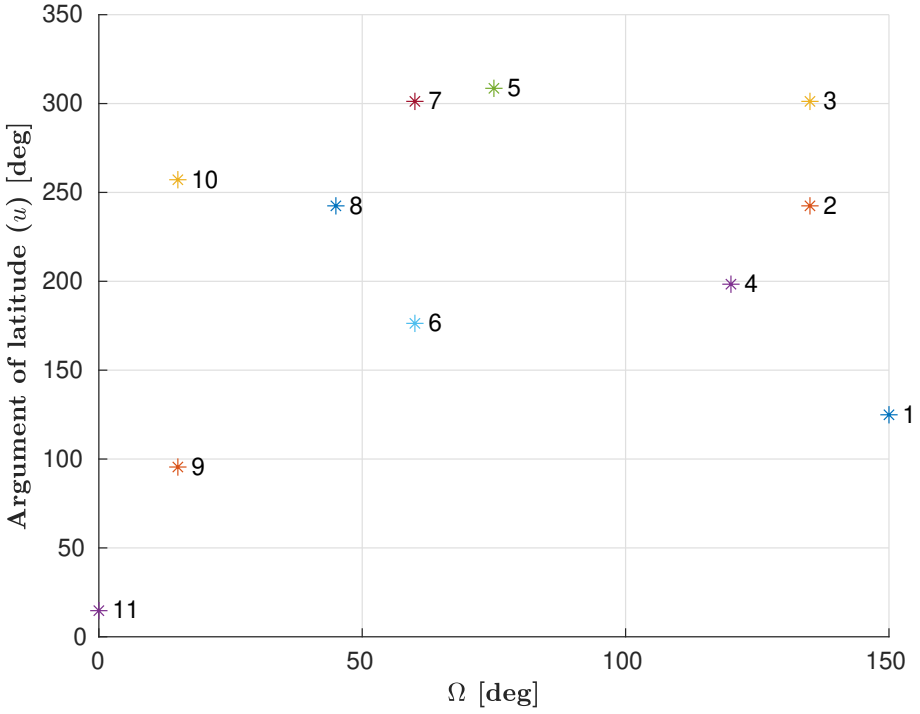


Figure 4.9: Branch and cut solution for the *OneWeb* constellation, for scenario 1

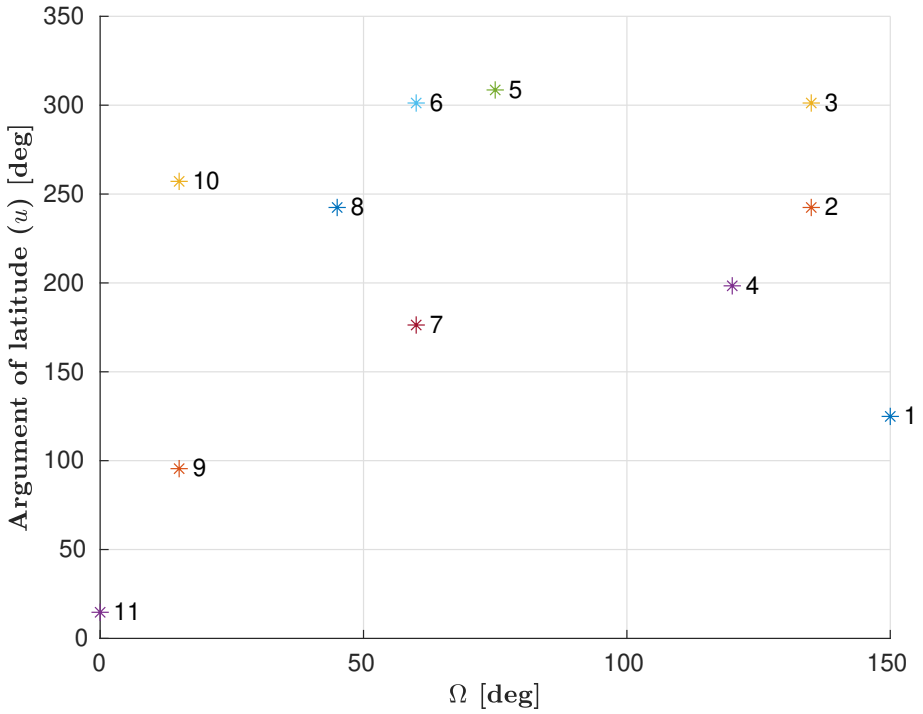


Figure 4.10: Nearest Neighbour solution for the *OneWeb* constellation, for scenario 1

Globalstar The distribution of the failures for the *Globalstar* constellation is presented in figure 4.11. The satellites are identified by an ID number inside the constellation, represented attached to the asterisks in figure 4.11. Simulation data and results are listed in table 4.5.

Table 4.5: Simulation data and results for *Globalstar* constellation for scenario 1

Parameter	Value
Constellation	Globalstar
Number of failed satellites	11
Brute force CPU time [seconds]	10.44
Branch and cut CPU time [seconds]	34.76
Nearest Neighbour CPU time [seconds]	0.000325
Brute force solution [months]	5.59
Branch and cut solution [months]	5.59
Nearest Neigbour solution [months]	5.59

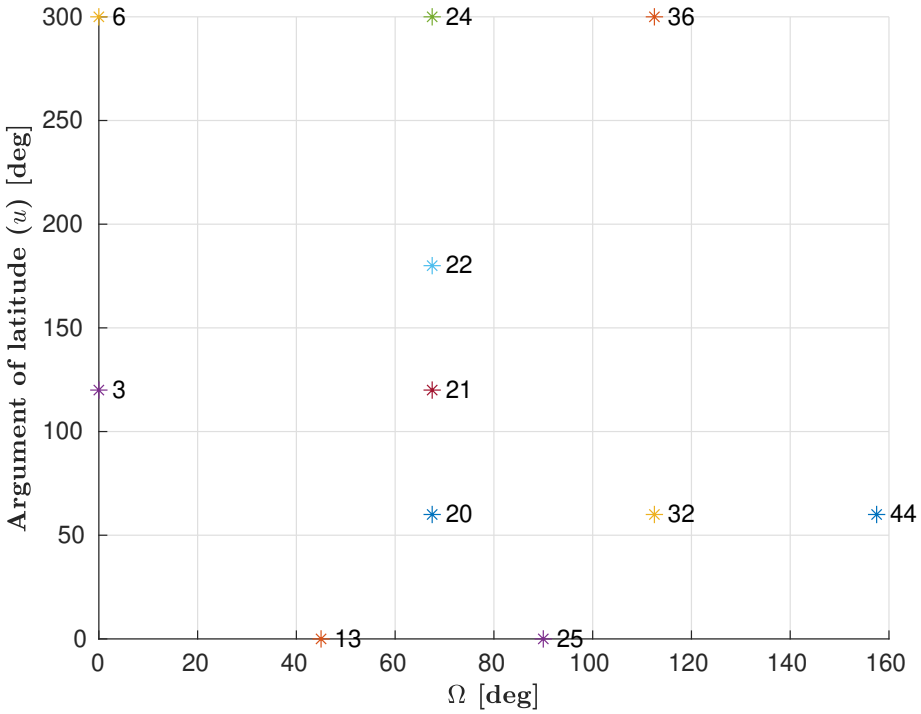


Figure 4.11: Distribution of the failures for the *Globalstar* constellation, for scenario 1

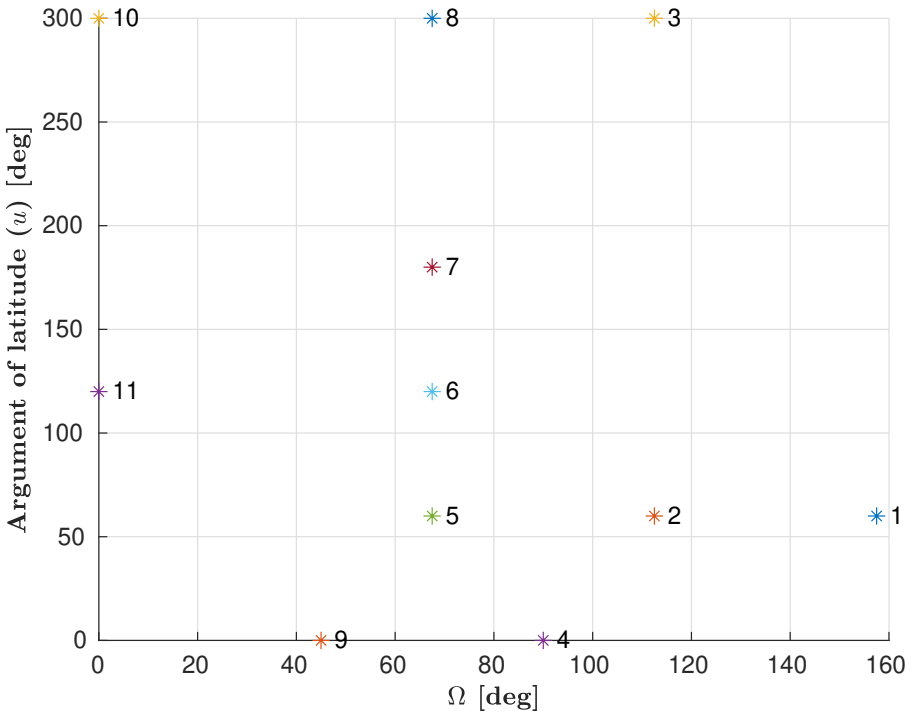


Figure 4.12: Brute force solution for the *Globalstar* constellation, for scenario 1

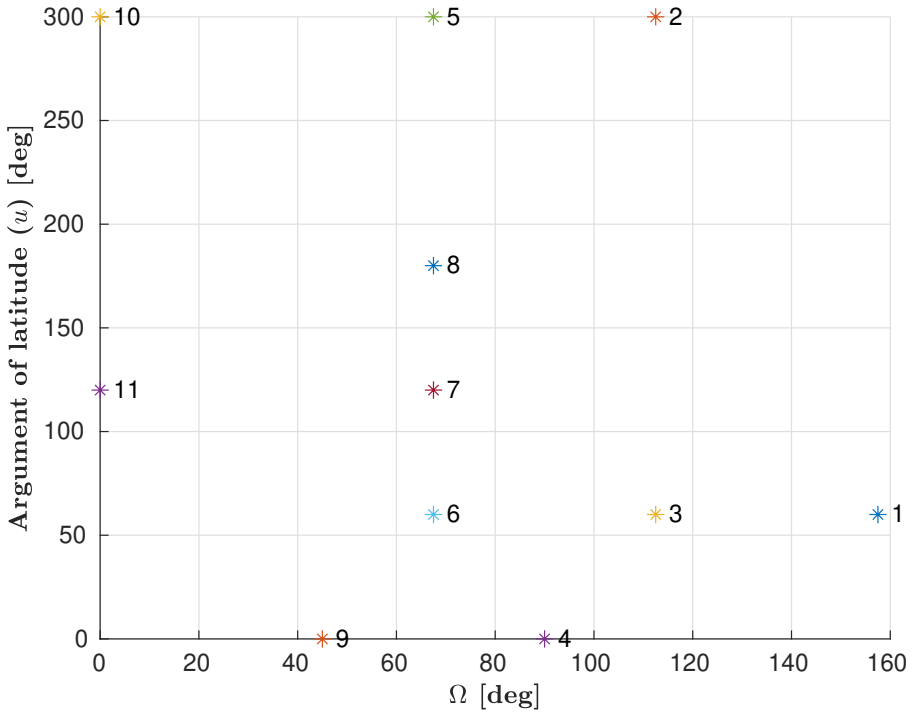


Figure 4.13: Branch and cut solution for the *Globalstar* constellation, for scenario 1

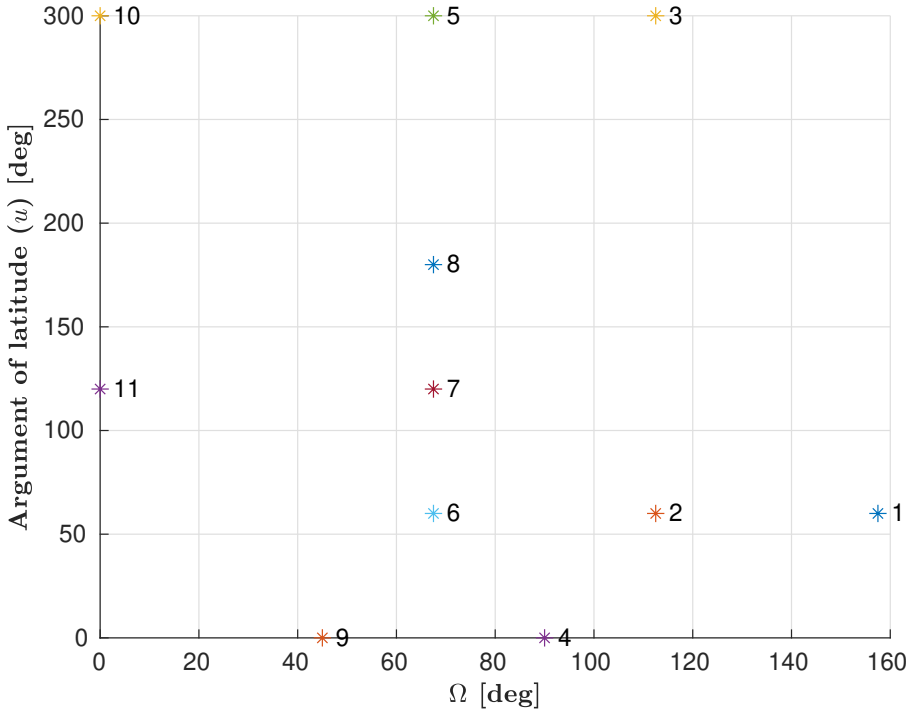


Figure 4.14: Nearest Neighbour solution for the *Globalstar* constellation, for scenario 1

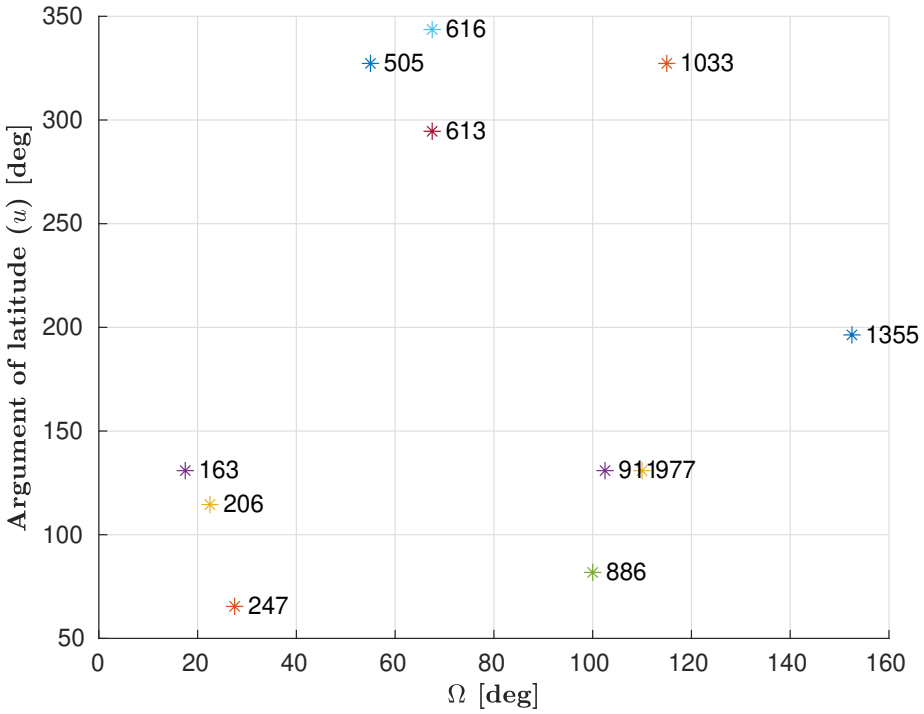


Figure 4.15: Distribution of the failures for the *Starlink* constellation, for scenario 1

Starlink The distribution of the failures for the *Starlink* constellation is presented in figure 4.15. The satellites are identified by an ID number inside the constellation, represented attached to the asterisks in figure 4.15. Simulation data and results are listed in table 4.6.

Table 4.6: Simulation data and results for *Starlink* constellation for scenario 1

Parameter	Value
Constellation	Starlink
Number of failed satellites	11
Brute force CPU time [seconds]	9.35
Branch and cut CPU time [seconds]	43.38
Nearest Neighbour CPU time [seconds]	0.000343
Brute force solution [years]	1.1955
Branch and cut solution [years]	1.1955
Nearest Neigbour solution [years]	1.1955

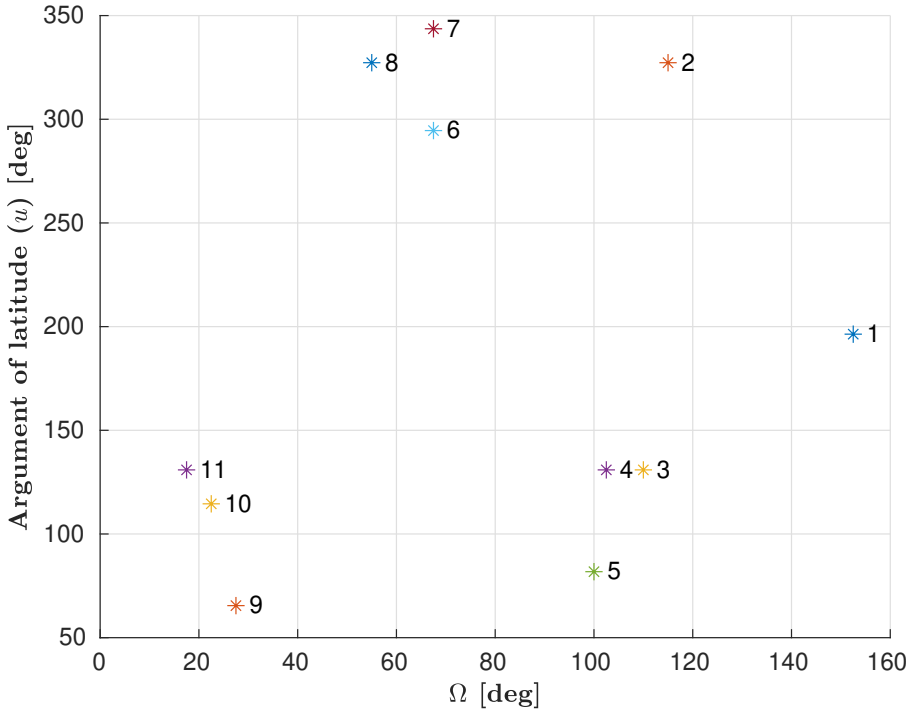


Figure 4.16: Brute force solution for the *Starlink* constellation, for scenario 1

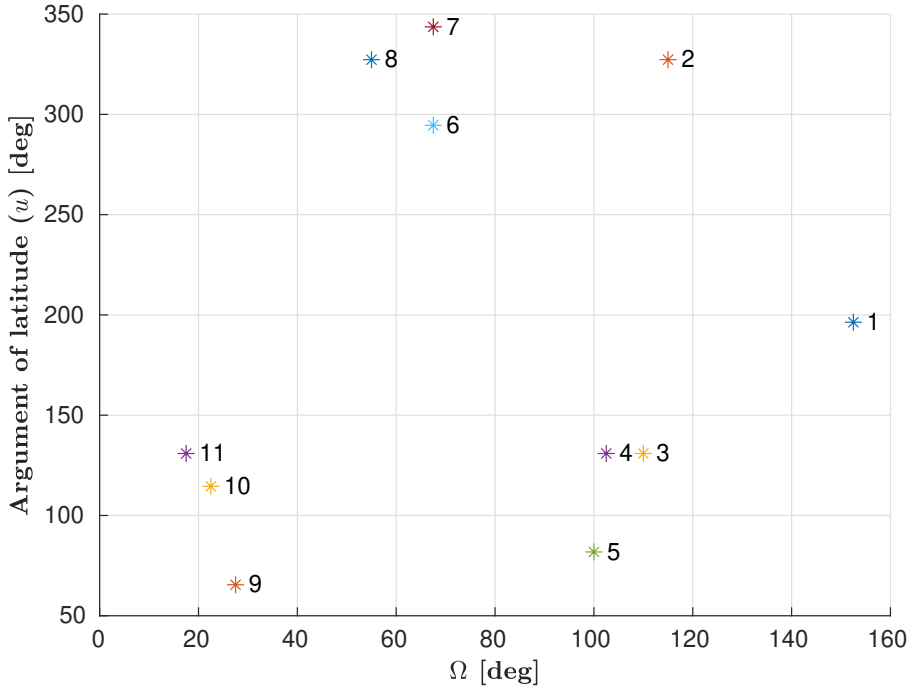


Figure 4.17: Branch and cut solution for the *Starlink* constellation, for scenario 1

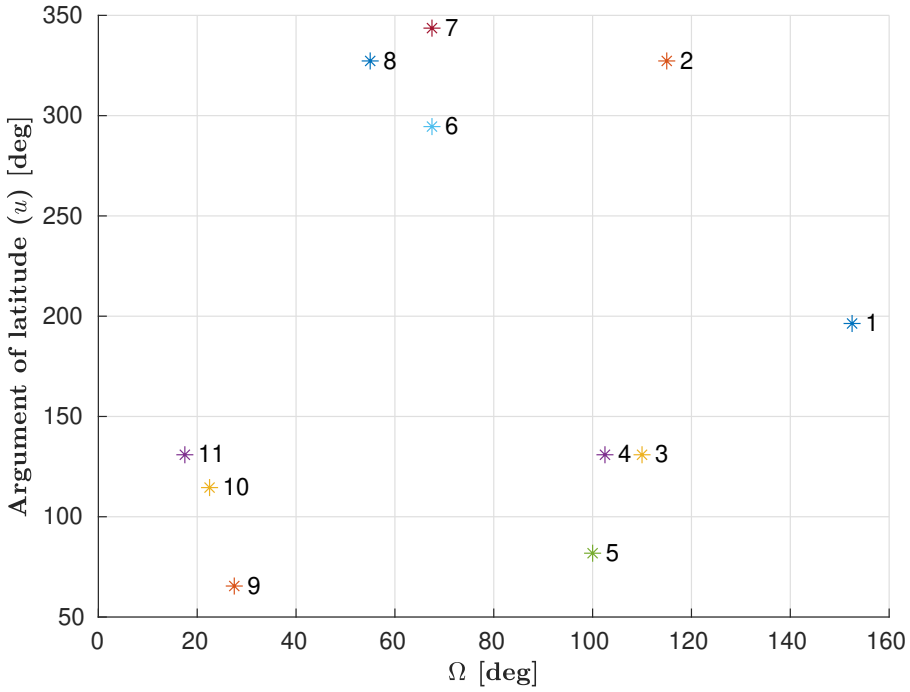


Figure 4.18: Nearest Neighbour solution for the *Starlink* constellation, for scenario 1

Scenario 2 - Failures randomly distributed in three orbital planes

OneWeb The distribution of the failures for the *OneWeb* constellation is presented in figure 4.19. The satellites are identified by an ID number inside the constellation, represented attached to the asterisks in figure 4.19. Simulation data and results are listed in table 4.7.

Table 4.7: Simulation data and results for *OneWeb* constellation for scenario 2

Parameter	Value
Constellation	OneWeb
Number of failed satellites	11
Brute force CPU time [seconds]	21.41
Branch and cut CPU time [seconds]	193.16
Nearest Neighbour CPU time [seconds]	0.0197
Brute force solution [years]	1.6480
Branch and cut solution [years]	1.6480
Nearest Neigbour solution [years]	1.6488

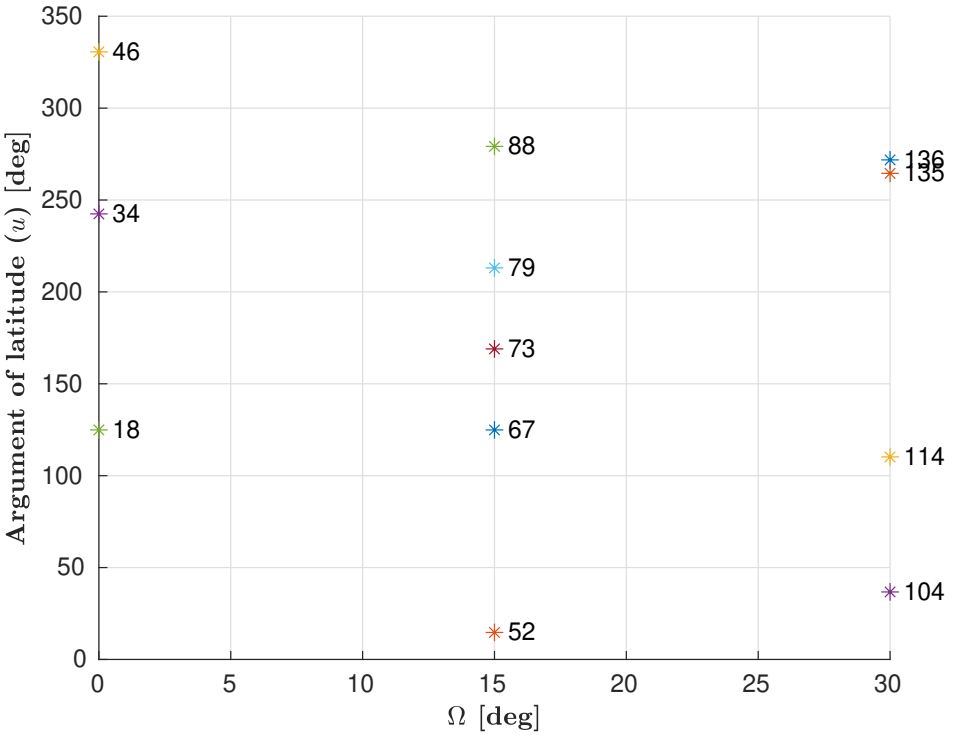


Figure 4.19: Distribution of the failures for the *OneWeb* constellation, for scenario 2

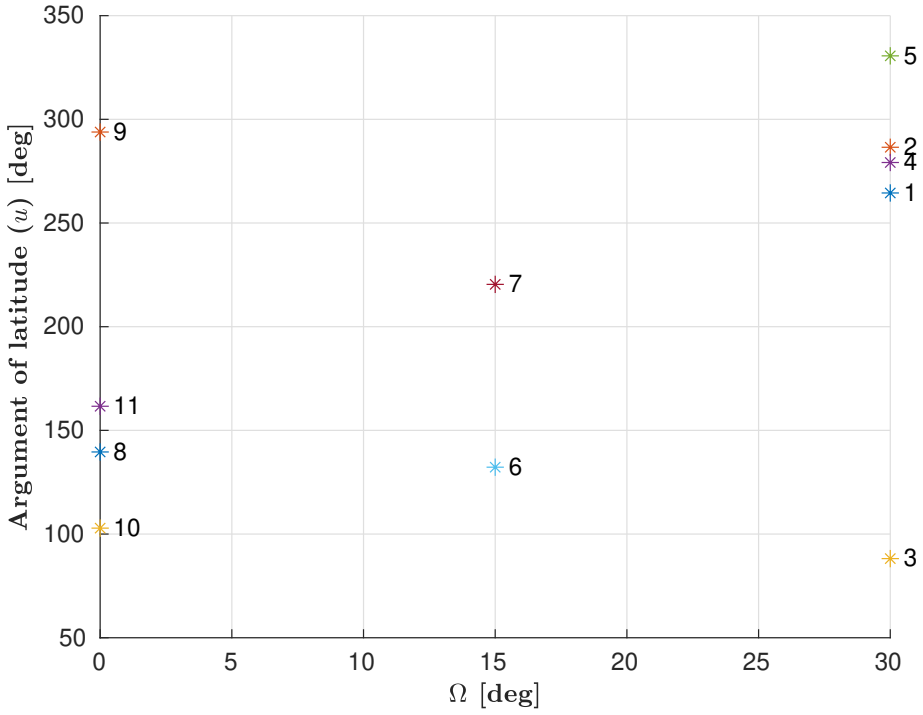


Figure 4.20: Brute force solution for the *OneWeb* constellation, for scenario 2

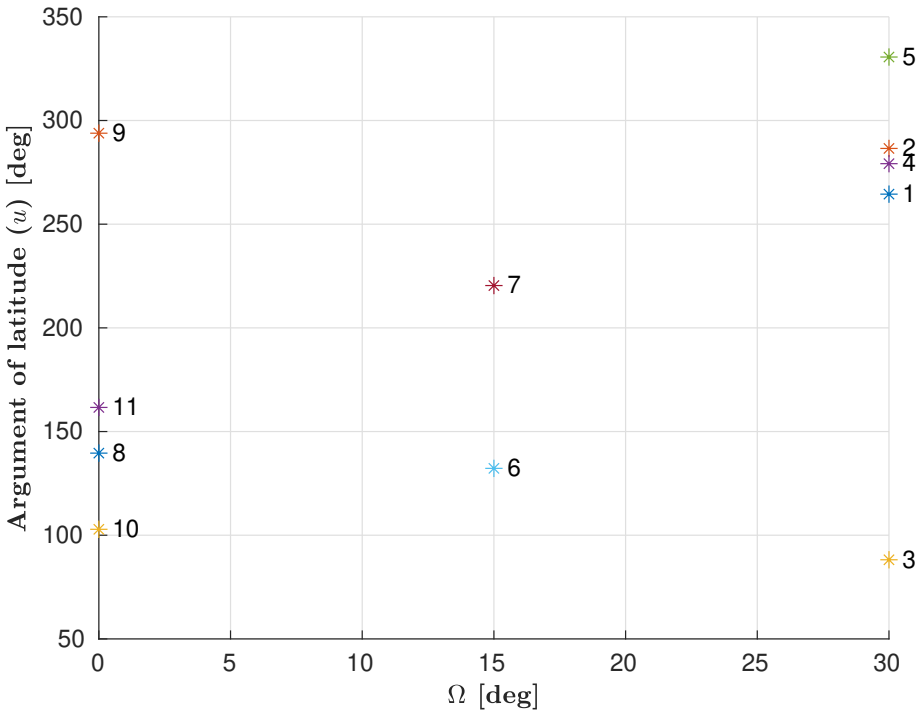


Figure 4.21: Branch and cut solution for the *OneWeb* constellation, for scenario 2

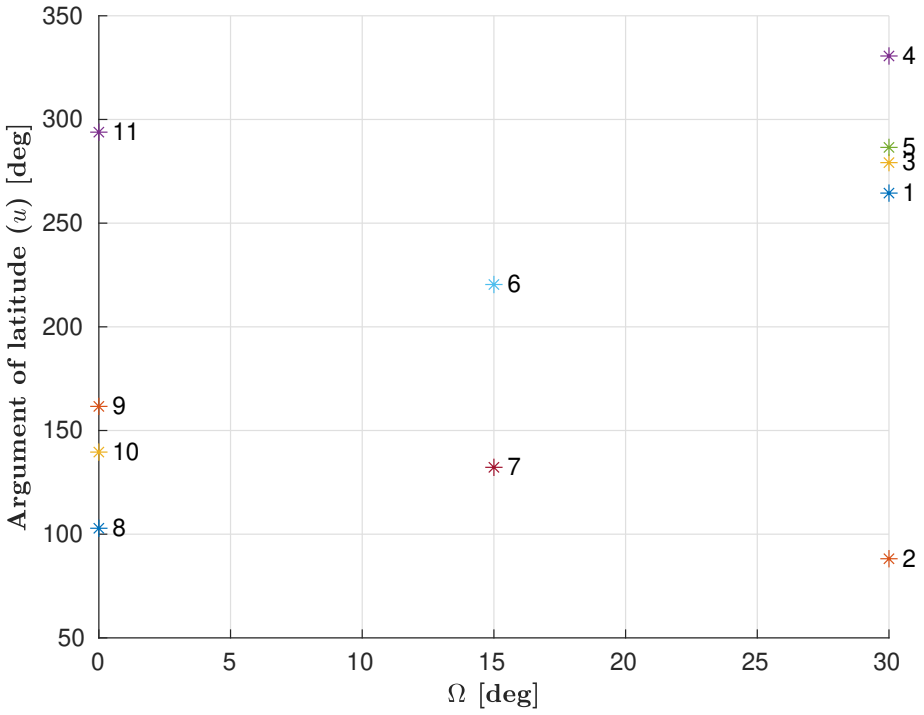


Figure 4.22: Nearest Neighbour solution for the *OneWeb* constellation, for scenario 2

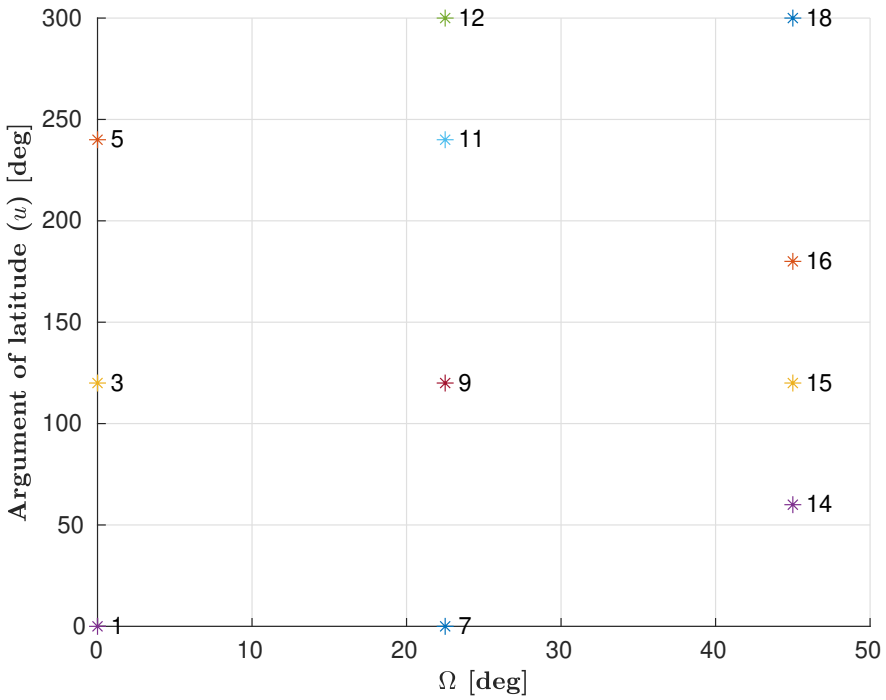


Figure 4.23: Distribution of the failures for the *Globalstar* constellation, for scenario 2

Globalstar The distribution of the failures for the *Globalstar* constellation is presented in figure 4.23. The satellites are identified by an ID number inside the constellation, represented attached to the asterisks in figure 4.23. Simulation data and results are listed in table 4.8.

Table 4.8: Simulation data and results for *Globalstar* constellation for scenario 2

Parameter	Value
Constellation	Globalstar
Number of failed satellites	11
Brute force CPU time [seconds]	10.17
Branch and cut CPU time [seconds]	148.72
Nearest Neighbour CPU time [seconds]	0.0016
Brute force solution [years]	0.1147
Branch and cut solution [years]	0.1147
Nearest Neigbour solution [years]	0.1147

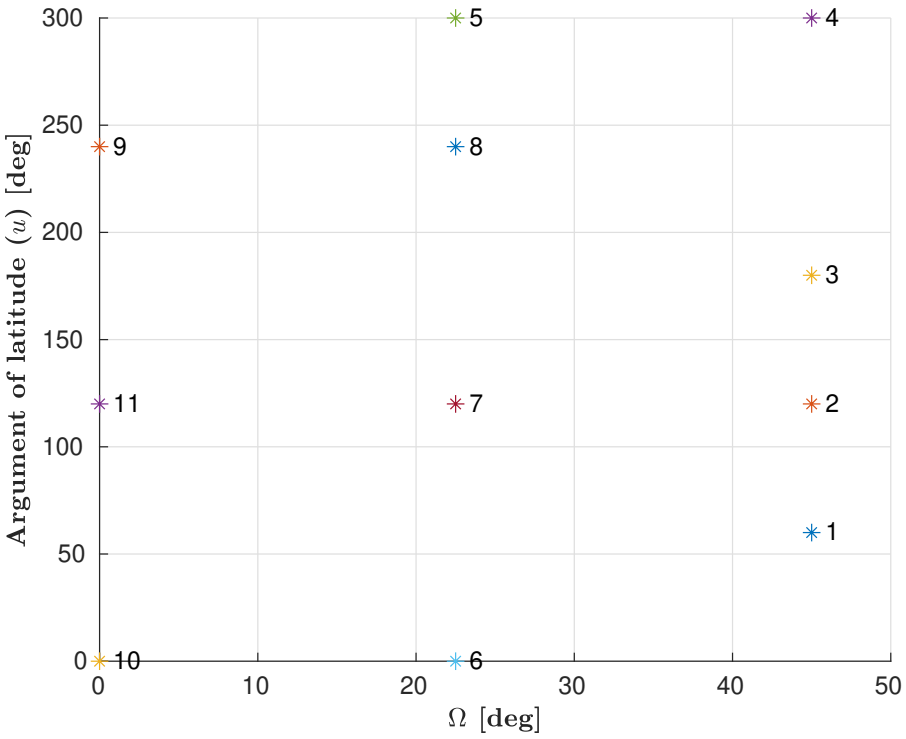


Figure 4.24: Brute force solution for the *Globalstar* constellation, for scenario 2

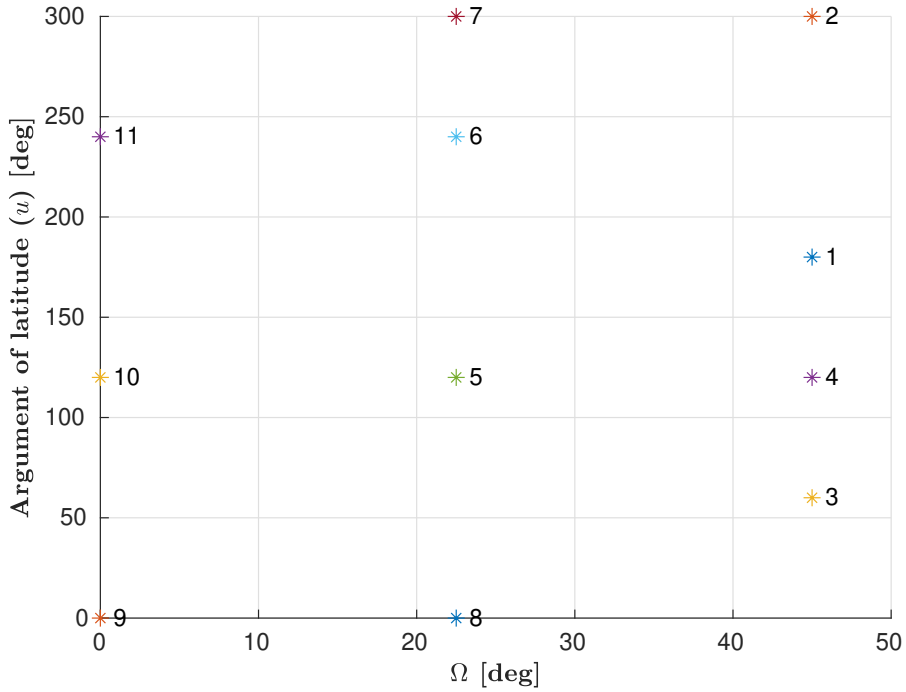


Figure 4.25: Branch and cut solution for the *Globalstar* constellation, for scenario 2

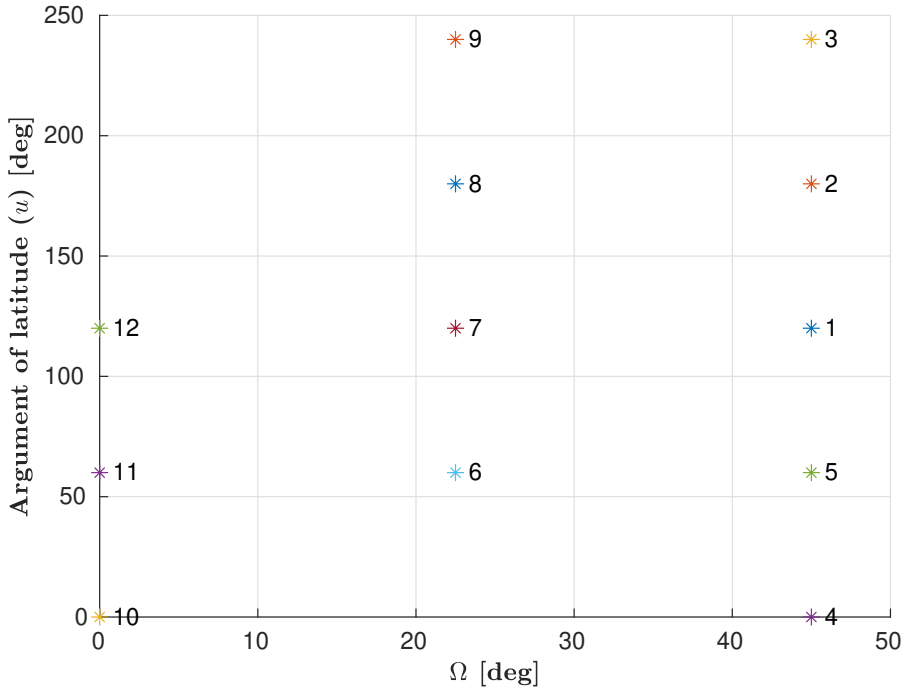


Figure 4.26: Nearest Neighbour solution for the *Globalstar* constellation, for scenario 2

Starlink The distribution of the failures for the *Starlink* constellation is presented in figure 4.27. The satellites are identified by an ID number inside the constellation, represented attached to the asterisks in figure 4.27. Simulation data and results are listed in table 4.9.

Table 4.9: Simulation data and results for *Globalstar* constellation for scenario 2

Parameter	Value
Constellation	Starlink
Number of failed satellites	11
Brute force CPU time [seconds]	9.68
Branch and cut CPU time [seconds]	146.18
Nearest Neighbour CPU time [seconds]	0.000295
Brute force solution [days]	18.56
Branch and cut solution [days]	18.56
Nearest Neigbour solution [days]	18.69

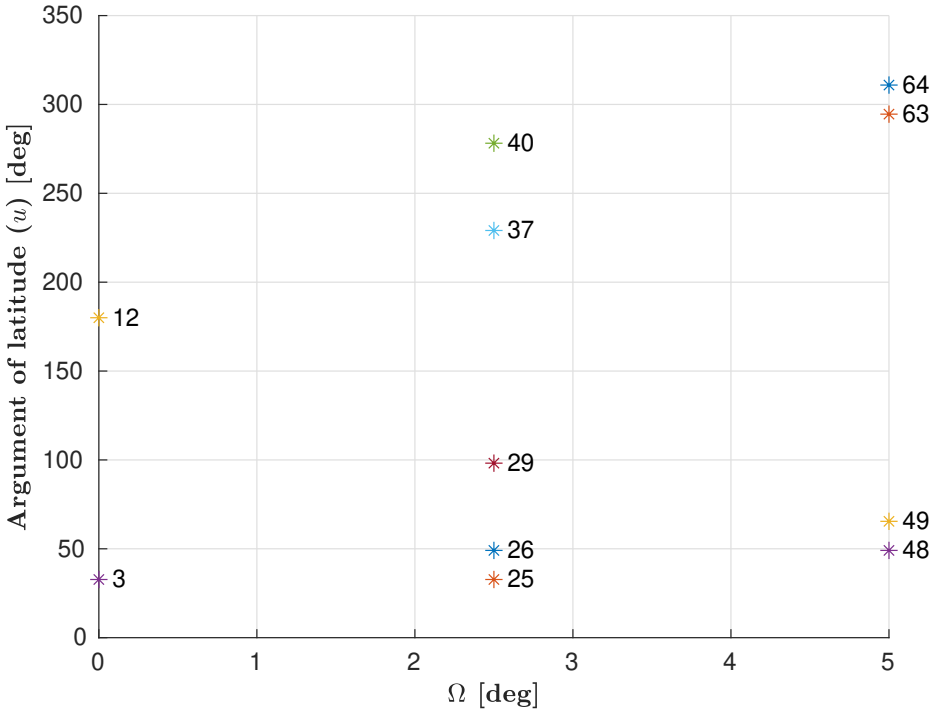


Figure 4.27: Distribution of the failures for the *Starlink* constellation, for scenario 2

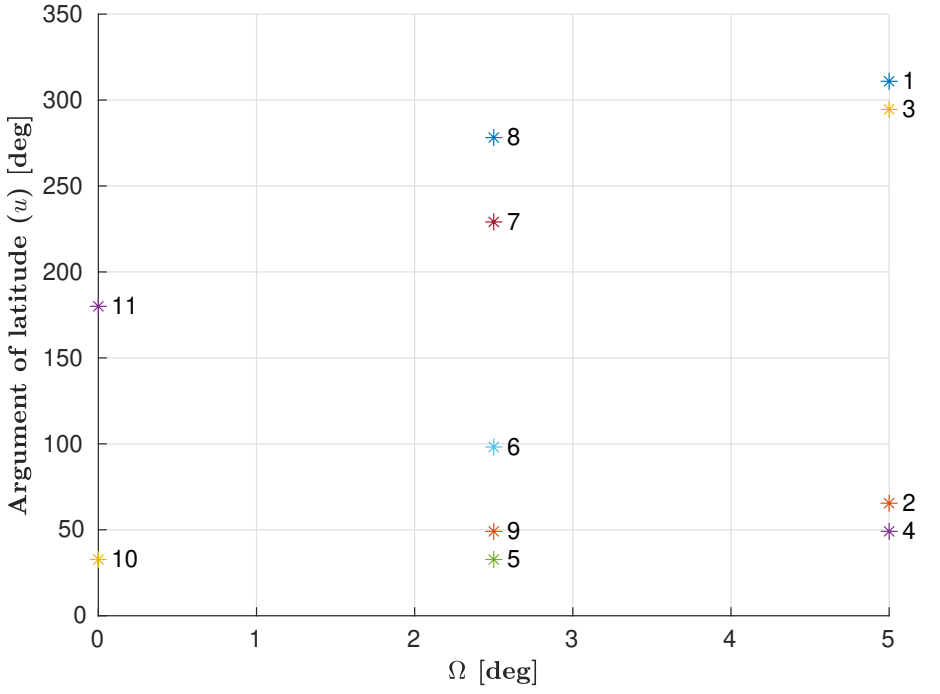


Figure 4.28: Brute force solution for the *Starlink* constellation, for scenario 2

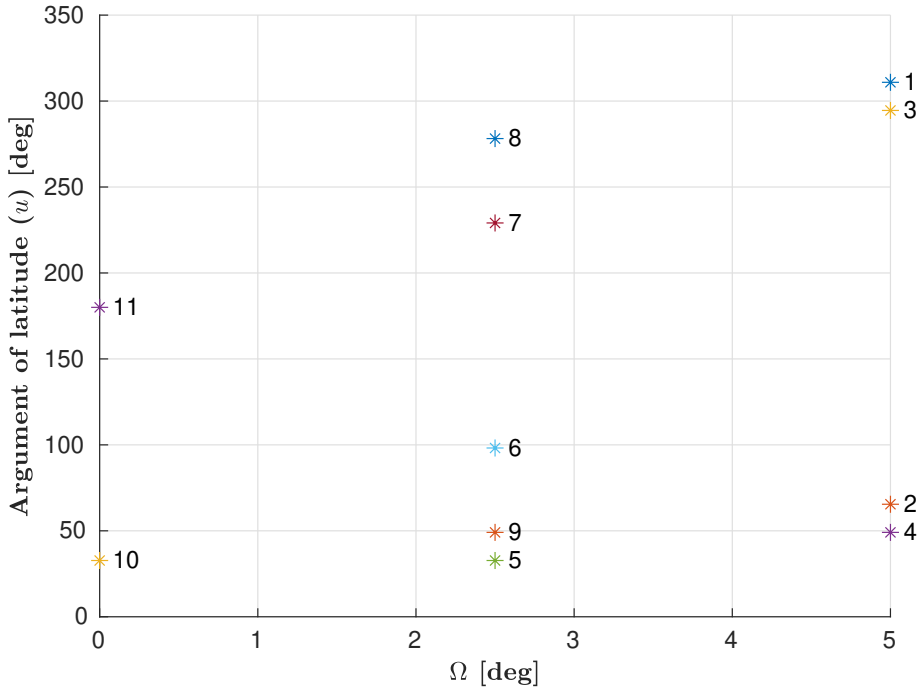


Figure 4.29: Branch and cut solution for the *Starlink* constellation, for scenario 2

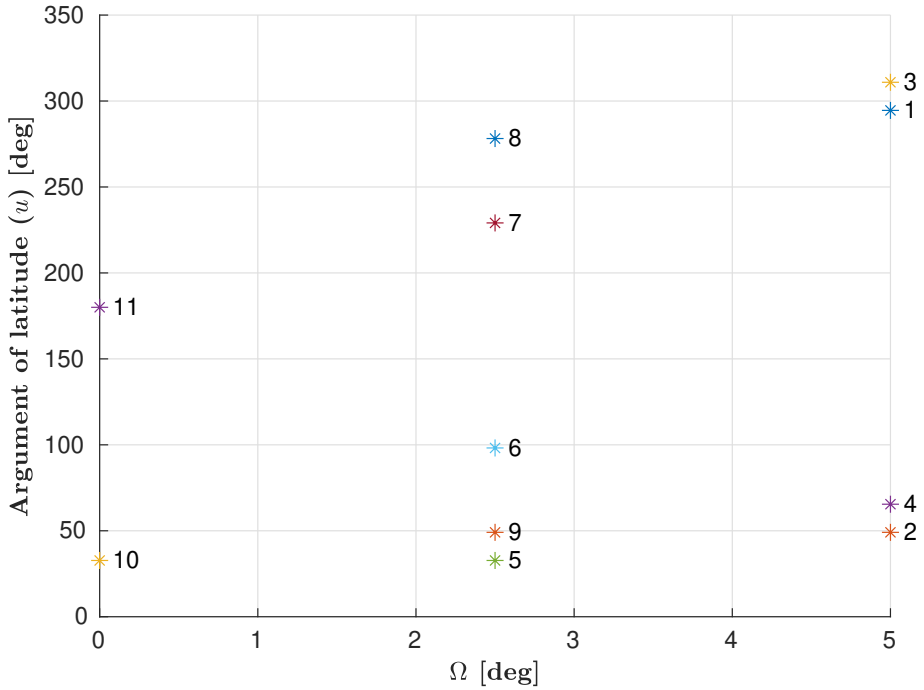


Figure 4.30: Nearest Neighbour solution for the *Starlink* constellation, for scenario 2

Monte Carlo simulations

To compare the performances of the three algorithms the Monte Carlo simulations are performed for the two scenarios. Tables 4.10 and 4.11 summarise the results.

Table 4.10: Monte Carlo for scenario 1 simulation - 100 samples

	OneWeb	*Globalstar*	*Starlink*
Number of dead satellites	10	10	10
Brute force mean CPU time [seconds]	0.6356	0.6350	0.6502
Branch and cut mean CPU time [seconds]	5.2431	5.8604	4.2113
Nearest Neigbour mean CPU time [seconds]	2.2469×10^{-4}	1.8841×10^{-4}	$1.9286 \times 10{-}4$
Branch and cut maximum percentage error [%]	0	0	0
Nearest Neigbour maximum percentage error [%]	0.05643	0.4884	0.5017

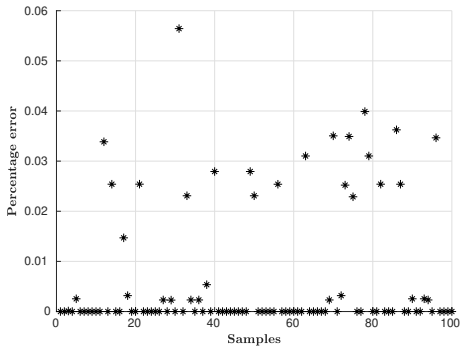
Table 4.11: Monte Carlo for scenario 2 simulation - 100 samples

	OneWeb	*Globalstar*	*Starlink*
Number of dead satellites	10	10	10
Brute force mean CPU time [seconds]	0.6439	0.6632	0.6451
Branch and cut mean CPU time [seconds]	11.0941	11.2890	11.3926
Nearest Neigbour mean CPU time [seconds]	1.9187×10^{-4}	1.9289×10^{-4}	$1.9238 \times 10{-}4$
Branch and cut maximum percentage error [%]	0	0	0
Nearest Neigbour maximum percentage error [%]	0.1505	0.8211	23.44

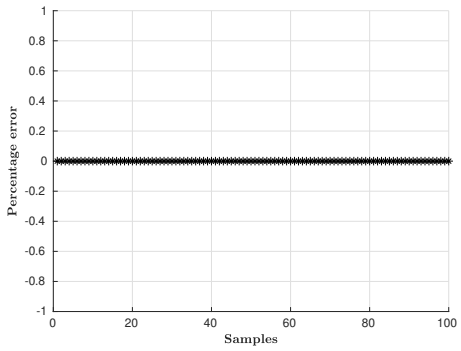
Note that the mean CPU time for each algorithm is calculated as the average of all the k CPU times, where k is the number of samples for the Monte Carlo simulation. The maximum percentage error compares the solution by the NNA or branch and cut method with the exact solution by the brute force method. It is expressed in this way:

$$e = max \left| \frac{y_{bf} - y_i}{y_{bf}} \right| \tag{4.11}$$

where y_{bf} is the solution found with the brute force algorithm and y_i is the solution found with NNA or branch and cut algorithm.

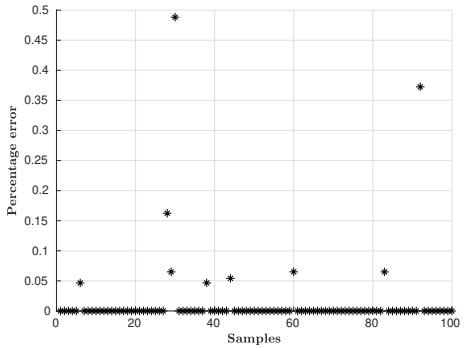


(a) *OneWeb*: NNA maximum percentage error

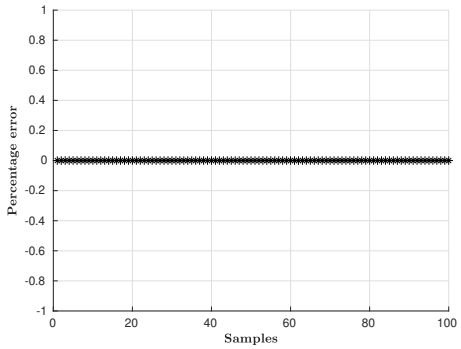


(b) *OneWeb*: Branch and cut maximum percentage error

Figure 4.31: Monte Carlo for scenario 1 simulation - *OneWeb*: NNA and Branch and cut quality performances

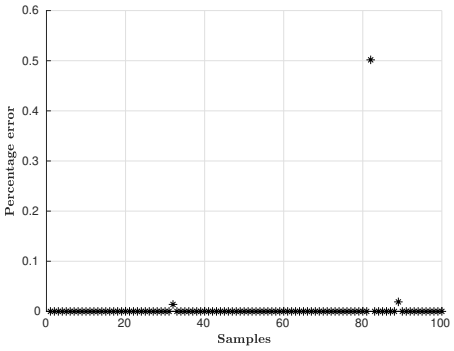


(a) *Globalstar*: NNA maximum percentage error

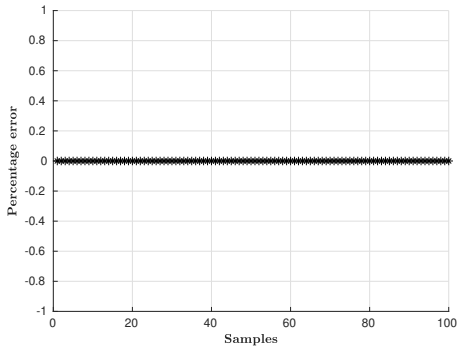


(b) *Globalstar*: Branch and cut maximum percentage error

Figure 4.32: Monte Carlo for scenario 1 simulation - *Globalstar*: NNA and Branch and cut quality performances

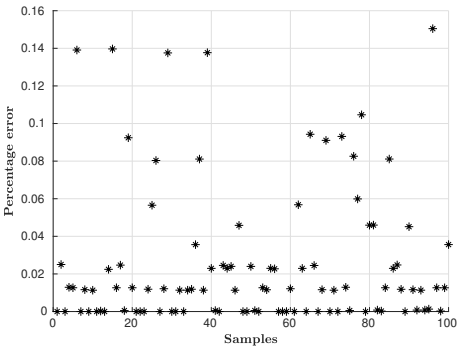


(a) *Starlink*: NNA maximum percentage error

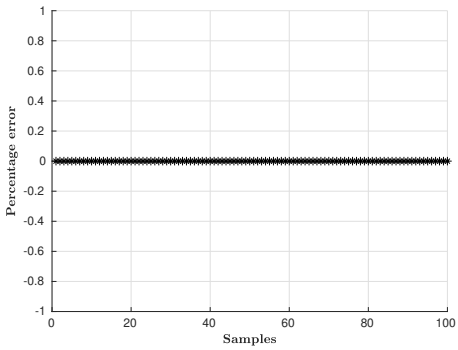


(b) *Starlink*: Branch and cut maximum percentage error

Figure 4.33: Monte Carlo for scenario 1 simulation - *Starlink*: NNA and Branch and cut quality performances

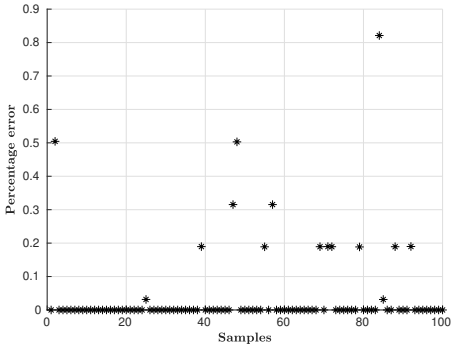


(a) *OneWeb*: NNA maximum percentage error

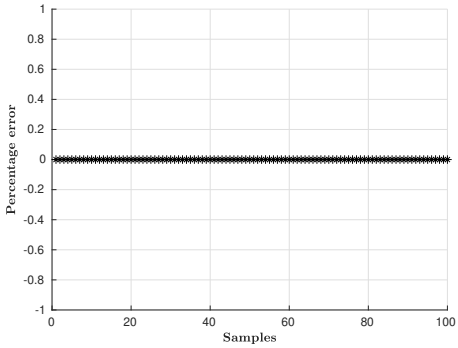


(b) *OneWeb*: Branch and cut maximum percentage error

Figure 4.34: Monte Carlo for scenario 2 simulation - *OneWeb*: NNA and Branch and cut quality performances

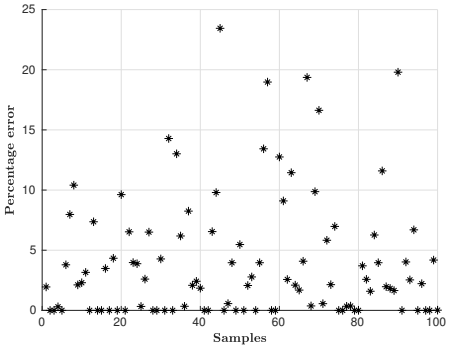


(a) *Globalstar*: NNA maximum percentage error

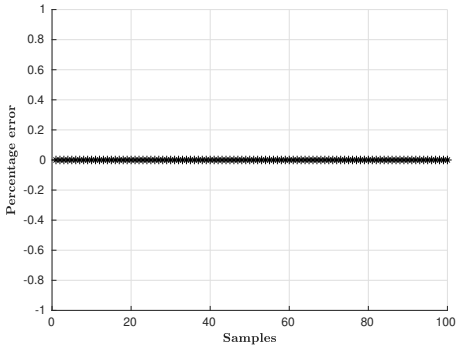


(b) *Globalstar*: Branch and cut maximum percentage error

Figure 4.35: Monte Carlo for scenario 2 simulation - *Globalstar*: NNA and Branch and cut quality performances



(a) *Starlink*: NNA maximum percentage error



(b) *Starlink*: Branch and cut maximum percentage error

Figure 4.36: Monte Carlo for scenario 2 simulation - *Starlink*: NNA and Branch and cut quality performances

Results discussion

Three different methods have been used for the optimal design of the ADR service. The main difference among all the algorithms is the computational time. *Nearest Neighbour* algorithm is the fastest one in all the situations, and it is slightly affected by the increasing number of vertices or by the mission scenarios. In the two scenarios the order of magnitude of the CPU time is the same, much lower than the others. The slower method is represented by the *Branch and cut* algorithm. Despite the fact that parallel computation is exploited, it is always slower than the *Brute force* approach. Because the maximum number of vertices analysed was relatively small ($n = 11$), checking all the branches before evaluating or pruning them slows down the code, which is a drawback of this approach. Moreover, the function generating the branches one by one, is much more slower than the built-in MATLAB function *perms*, which creates the branches much faster. Although these limitations, the *branch and cut* algorithm developed for this work, allows the problem to be solved even for a large value of n, while it is impossible for the *brute force* algorithm to face problems with a number of vertices larger than 11, due to memory time limitations. Furthermore, the performances of the *branch and cut* approach can significantly be improved exploiting workstation with multiple processors. The second aspect analysed is related to the quality of the solution obtained by the three approaches. Obviously, when it is feasible to apply the *brute force* method the global optimum is always found. Moreover, the *branch and cut* algorithm used in the simulations have a percentage of success of 100% when compared to the known optimum solutions. Thus, a good percentage of success is expected even with large-scale problems. The most interesting results came from the *Near Neighbour* algorithm. Although it is well known that for complicated problems this approach is not reliable, in the case study analysed, the quality of the solutions generated are very impressive. In the Monte Carlo simulation for scenario 1 the maximum percentage of error with respect to known optima was 0.5017%. When the density of the failures increases in scenario 2, NNA shows worst performances. The maximum percentage error related to *Starlink* constellation was 23.44%. Looking at figures 4.33a and 4.36a it is possible to observe that this values are represented by isolated points and do not follow a specific trend. Thus, it is possible to say that:

- The brute force method can always find the optimal solution because it checks all the possible paths. However, it does not suit for large-scale problems due to the high computational time and the demanding requirement for memory.

- The branch and cut method can cope with the drawback of the brute force method in terms of the demanding memory requirement, although it takes more computational time due to a pruning process. In addition, the branch and cut method can also find the optimal solution.

- The nearest neighbour algorithm reaches the optimal or close optimal solution in most cases when the failures density is relatively small. Moreover, the computational time is much lower than the other two approaches, since a lower number of operations need to be performed by the algorithm.

Chapter 5

Conslusions

The aim of this study was to analyse two possible architectures that have the application potential for the ADR services applied to large constellations. The first architecture makes use of two different types of vehicle to accomplish the mission. The main vehicle named *mother ship* is in charge of reaching the failed satellites in the constellation. Then, a de-orbiting *kit* is attached to the dead spacecraft to transfer it towards a disposal orbit. The disposal orbit is designed to drive the failed satellite towards an atmospheric re-entry. The second architecture exploits a single vehicle named *chaser*, which is in charge of reaching and transferring to the disposal orbit the failed satellites.
A mission analysis has been conducted to check the performances of the first architecture in terms of velocity change and mission time. Two scenarios has been taken into consideration, to study the feasibility of the mission depending on the number of planes each *mother ship* is in charge of. Results show that the feasibility of the mission architecture strongly depends on the number of *mother ship* involved. In the first scenario (ADR1) the mission requirements were totally satisfied, while for the second scenario where the number of orbital planes is a half of the first scenario, the total mission time exceeded the maximum lifetime of the *mother ship*. Indeed, most of the time is spent by the *mother ships* during the drifting phase, when they transfer from the first orbital plane towards the second one.
The second architecture has been analysed within the framework of the TSP. An optimal strategy to minimise the total mission time has been developed by exploiting three different optimisation methods: brute force, branch and cut, and nearest neighbour. A branch and cut algorithm has been developed based on the popular branch and bound method. It has been compared to brute force approach to check the quality of the solution found and the relative computational time. It has been observed that the branch and cut algorithm can overcome the limitations of the brute force method in terms of required memory, although an increased computational time. The nearest neighbour algorithm is the one requiring less computational time, while the quality of the solution decreases as the failure density increases.
The performances of the three algorithms have been compared. Implementing the three algorithms to different case studies, it has been demonstrated that the total time required to reach the same quantity of dead objects, depends strongly on the type of satellite constellation considered.

Bibliography

[1] *Astroscale Official Website*. URL: https://astroscale.com/missions/active-debris-removal-adr/.

[2] Camilla Colombo. "Planetary Orbital Dynamics (PlanODyn) suite for long term propagation in perturbed environment". In: *6th International Conference on Astrodynamics Tools and Techniques (ICATT)*. 2016, pp. 14–17.

[3] A David. "Vallado. Fundamentals of Astrodynamics and Applications". In: (2013).

[4] *End-of-Life Service by Astroscale (ELSA)*. URL: https://astroscale.com/missions/elsa-d/.

[5] *ESA, Space Engineering and Technology*. URL: http://www.esa.int/Enabling_Support/Space_Engineering_Technology/The_Kessler_Effect_and_how_to_stop_it.

[6] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[7] N Johnson. "USA Space Debris Environment, Operations and Policy Updates". In: *48th Session of the Scientific and Technical Subcommittee Committee on the Peaceful Uses of Outer Space, United Nations* (2011), pp. 7–18.

[8] Andrew T Magis and Nathan D Price. "The top-scoring 'N'algorithm: a generalized relative expression classification method from small numbers of biomolecules". In: *BMC bioinformatics* 13.1 (2012), pp. 1–11.

[9] *OneWeb Minisatellite Constellation for Global Internet Service*. URL: https://directory.eoportal.org/web/eoportal/satellite-missions/o/oneweb.

[10] *Starlink Satellite Constellation of SpaceX*. URL: https://directory.eoportal.org/web/eoportal/satellite-missions/s/starlink.