POLITECNICO DI TORINO

Masters Degree course in Aerospace Engineering

a.y. 2020/2021

December 2021

Mission optimization for LEO space debris removal using graph theory

Supervisor:
Prof. Lorenzo Casalino

Candidate:
Vittorio Friolotto
s274271

Abstract

"The Kessler Run" is a competition in which participants would have to find the best way to reach and remove 123 debris in LEO orbit with a minimum cost.
The starting point of the work was the preliminary solution of a previous project that aimed to find one solution to the mentioned competition. Such solution involves the use of graph theory to develop the starting sequences of the solution. Those sequences are, however, uncompleted and require additional processing in order to get all the 123 debris.
The goal of the work was to optimize such phase, already featured in the first finished project but needing improvements, in order to obtain better solutions. This purpose has been achieved via the implementation of 3 new methods of sequence-filling: a *direct debris insertion* method, that finds the possible spots in which new debris can be added to the sequence, a *replacement-insertion* method, that finds the best substitution of left debris with placed debris that can lead to a consequent additional placement, and a *generated-spot insertion* method, that creates, where possible, enough space to place a new debris in a sequence. All the starting solution, as well as the newly developed methods are implemented in MATLAB®. New results have been found and are discussed at the end of the paper in comparison with the starting solution, highlighting advantages and potential improvements.

Acknowledgements

Ringrazio il prof. Lorenzo Casalino per avermi seguito nel percorso di tesi, per la sua disponibilità ed i suoi preziosi consigli.
Ringrazio i miei colleghi per aver reso divertente quest'avventura importante.
Ringrazio la mia famiglia e i miei amici per il supporto durante questi anni.

In particolare, ringrazio mia mamma, per essere sempre stata un enorme sicurezza e un appoggio del quale non avrei mai potuto fare a meno.

Ringrazio mio papà, per essere sempre stato fiero di me e per aver sempre creduto in me, dandomi la motivazione di cui avevo bisogno.

Ringrazio mia nonna, che è sempre stata disposta a mettere me al primo posto e che non ha mai mancato di aiutarmi nel migliore dei modi.

Ringrazio mio nonno, per avermi aiutato a diventare chi sono e per essere sempre stato una grande ispirazione.

Ringrazio mio fratello, per aver sempre tifato per me e avermi aiutato ad affrontare le difficoltà con più leggerezza.

Ringrazio Claudia, per essermi stata vicina in ogni momento ed avermi sempre dato tutto il suo appoggio incondizionato.

Ringrazio me stesso, per non aver smesso di insistere e per i sacrifici che mi hanno portato fin qui.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Space debris

As of November 9th, 2021, ESA estimated a number of artificial space debris using statistical models of 36500 debris in orbit around the earth greater than 10 cm, 1 million between 1 cm and 10 cm and 330 million between 1 mm and 1 cm.
We refer to space debris as all the objects in space that don't serve any useful purpose, such as abandoned rocket stages and satellites, debris generated from collisions and micro fragments derived from propellants, waste of human activities and particles or bodies from earth and outer space of natural origins.
The most concentrated area for orbital debris is in the area of space within 2000 km of the Earth surface, as can also be appreciated from the figure that reports a view from GEO orbit of space debris, generated by NASA ODPO.

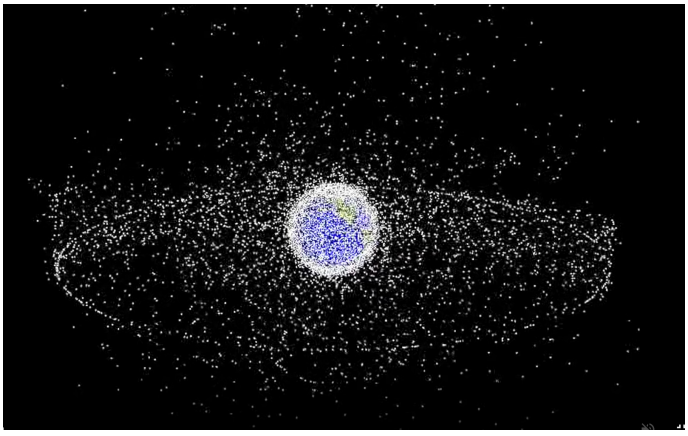

Figure 1.1. View of space debris from GEO, NASA ODPO

Space debris already existed before the start of the space age, but have assumed more importance in the last decades because of the hazard it may cause to human activities in space and even on Earth.
About 29600 debris objects are now catalogued and tracked by the Space Surveillance Network, and, looking at the numbers mentioned before, the risk of possible collisions with small particles is to be taken into account. The mean velocity for satellites orbiting in LEO is approximately between 7 to 8 km/s, this leads to collisions that can reach up to 15 km/s. Such numbers, even with small particles could lead to serious damage to unmanned and manned spacecraft: a number of space shuttle windows have been replaced because of the damage reported by small particle debris, that are, in fact, the highest risk to robotic spacecraft operating in LEO. As of today, Space Surveillance Network can track objects as small as 5 cm in LEO and as small as 1 meter in GEO.
The image represents the damage resulting from a collision at about 6.8 km/s of a sphere of 1.2 cm diameter with a 18 cm thick block of aluminium.

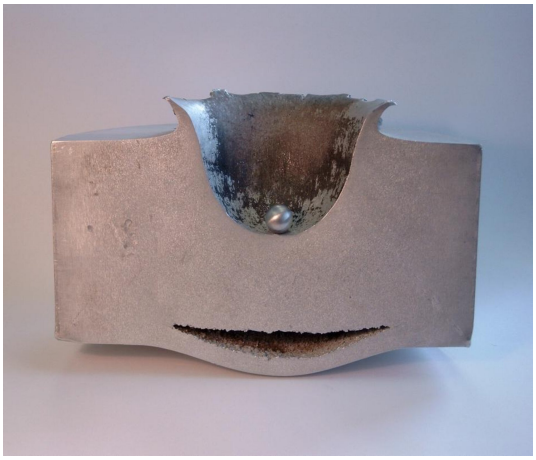

Figure 1.2. Impact of high velocity small object on aluminium block, ESA experiment

The number of debris kept growing from the first launch of an artificial satellite, Sputnik 1, into orbit on October 4th, 1957.
The growth is still happening and, without an active reaction by space authorities, will continue.
Due to new missions and new collisions, in the last decades, the growth of space debris has been estimated to be about 300 debris/year, as figure 1.3 shows.

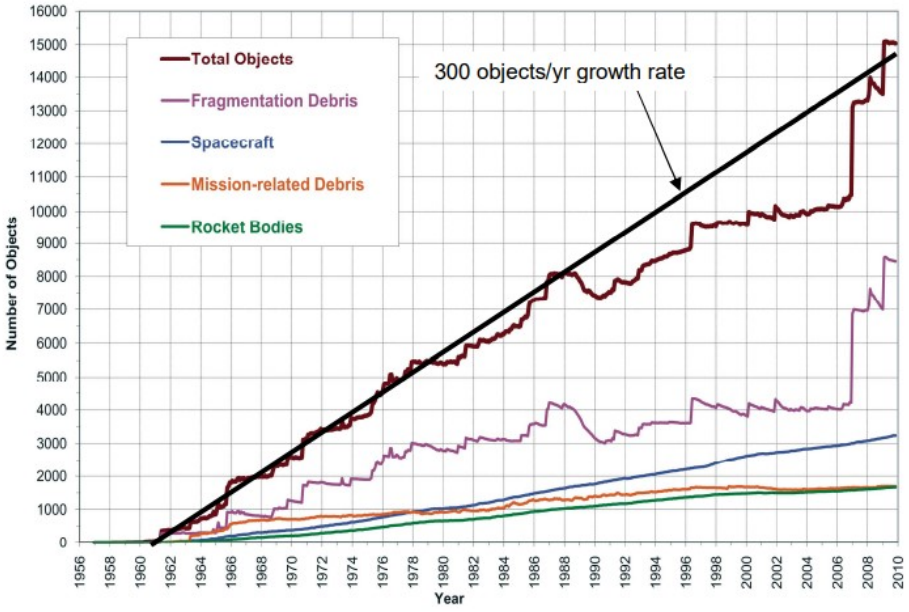

Figure 1.3. Debris trend from 1957 to 2010

Two big steps are identifiable in the total debris trend are due to two main events happened in 2007 and 2009:

- 2007: China's anti-satellite test, which destroyed a weather satellite, resulting in 3500 debris large enough to be tracked and many more small particles.

- 2009: Iridium satellite collision with an abandoned Russian spacecraft that resulted in 2300 traceable debris.

1.1.1 The Kessler Syndrome

The constant growth in the number of debris may lead to an exponential trend resulting in a debris belt around Earth.
In 1978 NASA Astrophysicist Donald J. Kessler proposed a scenario in which Earth would be surrounded by debris and, therefore, isolated from outer space. The theory, introduced later by John Gabbard from NORAD as "Kessler syndrome", predicts that the presence of numerous debris would lead to a cascade effect of new debris generated from already present debris collisions. This mechanism is also thought to be responsible of the generation of asteroids belts around planets in the solar system, the difference is that, if other planets' belts took billions of years to be created, in Kessler's theory, a much shorter time would be required because of the high number of debris around Earth.

In the paper a model describing the debris flux based on known earth satellites has been developed, and hyper velocity collisions have been examined to better understand debris generation mechanism concluding that by the year 2000 collisional breakups of satellites would become a new source of space debris and that, even without new launches, the process of exponential debris formation will occur in a shorter period of time than the asteroids belts observed around other planets.

1.1.2 Debris countermeasures

In order to protect human activity in space from orbiting debris, many countermeasures have been taken into account, that can be divided in 3 main groups: *mitigate human contribution to space debris*, *shield spacecraft and/or avoid debris* and *debris removal*.

Mitigate human contribution to space debris

In the *Orbital Debris Mitigation Standard Practices*, the US Government introduced 5 objectives and, for each, one or more standard practices to follow in order to reduce human impact on space debris:

1. *Control of debris released during normal operations*: where it's stated that spacecraft and upper stages should be designed in order to minimize debris release, and every debris over 5 mm in any dimension that remain in orbit for more than 25 years must be justified.

2. *Minimizing debris generated by accidental explosions*: the risk of accidental explosions should be demonstrated to be less than 0.1% and all on-board sources of stored energy that could cause additional risk should be safed or depleted when they are no longer required for mission operations.

3. *Selection of safe flight profile and operational configuration*: collision with debris larger than 10 cm must be ensured to have a probability of happening less that 0.1%, while collision with small debris, smaller than 1 cm with a probability of 1%.

4. *Post-mission disposal of space structures*: a spacecraft or an upper stage should be disposed, with a probability of success not less than 0.9 by: direct reentry in atmosphere, earth escape or storage in low populated orbits for at least 100 years, depending on the orbit.

5. *Clarification and additional standard practices for certain classes of space operations*: where guidelines are clarified for particular type of missions, such as: satellite constellations, cubesat, rendezvous operations, active debris removal and tether systems.

Shield spacecraft and/or avoid debris

Spacecraft are, in order to protect on-board instrumentation and the structure itself, equipped with debris shield, of various nature and thickness. The ISS, to name one, is equipped with numerous typologies of shield: 1.5 to 5 cm thick aluminium layers, 10 cm thick Kevlar and Nextel shields, for example, depending on the estimated exposure to potential damage.
However the shield solution may not be applicable for every type of debris that could collide with spacecraft, for instance, on the ISS, for the US module, shields can only be effective against collisions with debris smaller than 1 cm. For this reason, collision-avoiding techniques have been thought. For the ISS, NASA provided a set of guidelines, one of which is the so-called "pizza box" because of its shape: a 4 km $\times$ 50 km $\times$ 50 km box with the ISS at its center. Whenever a big enough debris is expected to be too close to the station, avoiding manoeuvres may be actuated: if the probability is 1 in 100000, the manoeuvre will take place only if it will not cause significant impact on the mission objectives, if the probability if 1 in 10000 the manoeuvre will take place if it will not cause additional hazard to the station and the crew members. Since 1999, the ISS has conducted 29 collision-avoiding manoeuvres, 3 in 2020.
An example of collision with a debris on the ISS is shown in Figure 1.4, the picture shows a hole on the robotic ISS arm Canadarm2.

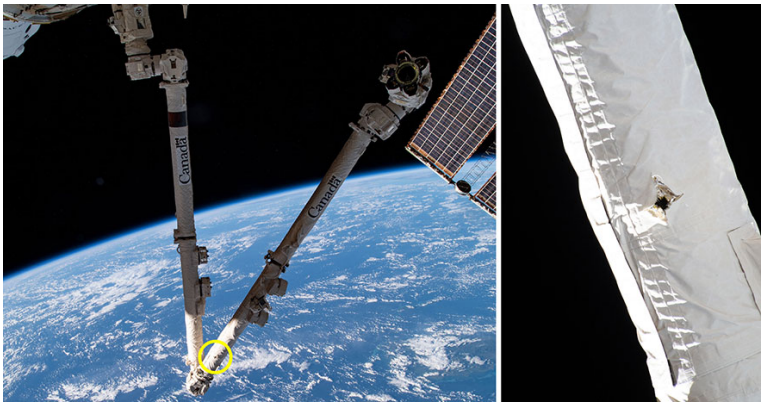

Figure 1.4. ISS Canadarm2 debris hole

Active debris removal

It has been estimated that, in order to mitigate the process of debris number growth, at least 5 debris should be removed every year. Many techniques of active debris removal have been studied and developed, to mention a few:

- **Propellantless systems**: in particular for large size debris (more than 10 cm), *lasers* via ablation or vaporization of the debris, *drag enhancement device*, usually inflatable device with the intent to increase atmospheric drag, *soar sails*, with the purpose of using solar radiation pressure in order to change the orbit of the debris or deorbit it, *electromagnetic tether*, that by interaction between earth magnetic field and a several km long conductive tether would generate enough force to redirect the debris to a deposit orbit or to deorbit and *momentum tethers* that uses the gravity gradient in order to redirect one end of the tether to another orbit.

- **Active** (and passive) **sweepers**: for medium size debris (between 5 mm and 10 cm), systems that rely on debris collision in order to remove them from overpopulated orbits. The sweepers, once deployed, will collide with debris that would be captured or even just decelerated by the collision, resulting in debris removal.

- **Deorbit kit**: this technique in particular is the one on which this paper will apply the methods developed. It consists in specifically developed kits that, once installed on debris that can have different dimensions, could relocate it on deposit orbits or deorbit it in atmosphere.

Chapter 2

Dynamic model and transfer cost evaluation

2.1 Dynamic model

Usually space debris dynamics are defined using a set of Two-line Elements (TLE) and SGP4 Propagator, however, since the accuracy of ephemerids degrades with time and TLE has to be updated regularly, a simplified propagation model has to be adopted.
The values of semi-major axis for the space debris of interest is sufficiently high, atmospheric drag can be neglected and only secular orbit perturbation due to Earth oblateness (related to harmonic terms of J_2) is considered.
Semi-major axis a, orbit eccentricity e and orbit inclination i are considered constant, while right ascension of ascending node Ω, argument of periapsis ω and mean anomaly M vary as

$$\frac{\mathrm{d}\Omega}{\mathrm{d}t} = -\frac{3}{2}\sqrt{\frac{\mu}{a^3}}\frac{J_2\cos i}{\left(1-e^2\right)^2}\left(\frac{r_E}{a}\right)^2$$

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{3}{4}\sqrt{\frac{\mu}{a^3}}\frac{J_2\left(5\cos^2 i-1\right)}{\left(1-e^2\right)^2}\left(\frac{r_E}{a}\right)^2$$

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \sqrt{\frac{\mu}{a^3}} + \frac{3}{4}\sqrt{\frac{\mu}{a^3}}\frac{J_2\left(3\cos^2 i-1\right)}{\left(1-e^2\right)^{3/2}}\left(\frac{r_E}{a}\right)^2$$

where μ is Earth's gravitational parameter and r_E is Earth's equatorial radius.
In case of low eccentricity, small changes of $a(\delta a)$ and $i(\delta i)$ will cause a deviation of RAAN rate $(\delta\dot{\Omega})$ that can be obtained by the derivative of $\dot{\Omega}$ to a and i:

$$\frac{\delta\dot{\Omega}}{\dot{\Omega}} = -\frac{2}{7}\frac{\delta a}{a} - \tan i\delta i$$

2.2 ΔV transfer cost evaluation

In order to calculate the cost of every transfer between debris, it has been used a simplified model based on the assumption that the RAAN of the chaser is close to the one of the target. In this way the transfer can take advantage of the J_2 perturbation, that modified the RAAN of bodies orbting around earth. This method can be applied to the studied problem since J_2 perturbation has appreciable effect on low orbits, at which all debris of GTOC 9 are placed.
The model gives 2 different solutions for 2 different problems: the first can be applicable only to transfers where there is no time constraint, and will be only mentioned, while the second provides solution that depends directly on the transfer duration, and will be the one used in the problem solution.

Without time constraint Based on the simplification of the Hohmann transfer cost for negligible radius change, that states that $\frac{\Delta v}{v} = 0.5\frac{\Delta r}{r}$, and adding an empirical relation to the eccentricity vector change, the model provides the solution

$$\frac{\Delta v}{v} = 0.5\sqrt{(\Delta a/a)^2 + \Delta i^2 + \Delta e^2)}$$

Where a is the mean semi-major axis value for the two orbits and v is the correspondent circular velocity.

Transfer time is fixed Every transfer is considered do be from debris k to debris $k+1$ and it is assumed that

$$\Omega_{k+1}(t) - \Omega_k \neq 0, \quad a_{k+1} - a_k \neq 0, \quad i_{k+1} - i_k \neq 0.$$

The three necessary velocity changes, denoted as x, y and z are

$$x = (\Omega_{k+1}(t) - \Omega_k(t)) \sin i_0 v_0$$

$$y = \frac{a_{k+1} - a_k}{2a_0} v_0$$

$$z = (i_{k+1} - i_k)\, v_0$$

where $a_0 = (a_{k+1}+a_k)/2$, $i_0 = (i_{k+1}+i_k)/2$ and $v_0 = \sqrt{\mu/a_0}$. In this approximation, 2 impulse velocity change are considered, in order to save fuel. The first impulse is written as

$$\Delta v_a = \sqrt{(s_x x)^2 + (s_y y)^2 + (s_z z)^2}$$

Since terms s_x, s_y, s_z have no constraints, semi-major axis and inclination changes may be larger than the actual difference in order to take more advantage of the J_2 effect. This occurs when RAAN change is too large, and time wouldn't be sufficient.

The changes in semi-major axis and inclination lead to a change in RAAN, that can be written as

$$\Delta x = m s_y y + n s_z z$$

where $m = (7\dot{\Omega}_0) \sin i_0 t$ and $n = (\dot{\Omega}_0 \tan i_0) \sin i_0 t$. Those two values are derived from the dynamic model approximation to which the transfer cost model refers. The second impulse can be written as

$$\Delta v_b = \sqrt{(x - s_x x - \Delta x)^2 + (y + s_y y)^2 + (z + s_z z)^2}$$

therefore, the total velocity change can be written as

$$\Delta v_a + \Delta v_b = \sqrt{(s_x x)^2 + (s_y y)^2 + (s_z z)^2} + \sqrt{(x - s_x x - \Delta x)^2 + (y + s_y y)^2 + (z + s_z z)^2}$$

Additional approximations are applied to the formula, ignoring small terms based on the first assumption that RAAN changes are small, allowing a differentiation with respect to s_x,s_y and s_z. The equations are therefore set equal to 0 in order to find minimum s_x, s_y and s_z that minimize velocity change, that are

$$s_x = \frac{2x + my + nz}{(4 + m^2 + n^2)x}$$

$$s_y = \frac{2mx - (4 + n^2)y + mnz}{(8 + 2m^2 + 2n^2)y}$$

$$s_z = \frac{2nx + mny - (4 + m^2)z}{(8 + 2m^2 + 2n^2)z}$$

For small eccentricity changes, the velocity change is

$$\Delta v_e = \frac{1}{2} v_0 \sqrt{\Delta e_y^2 + \Delta e_x^2}$$

where $e_y = e \sin \omega$ and $e_x = e \cos \omega$. An empirical eccentricity correction to the formula $\Delta v = \Delta v_a + \Delta v_b$ is introduced:

$$\Delta v' = \sqrt{\Delta v_a^2 + (0.5 \Delta v_e)^2} + \sqrt{\Delta v_b^2 + (0.5 \Delta v_e)^2}$$

assuming that the Δv change is divided equally between each impulse.

Chapter 3

Problem definition

3.1 GTOC9: "The Kessler Run"

The *Global Trajectory Optimization Competition* is an event that takes place every one or two years in which aerospace engineers and mathematician compete to find the best solutions to interplanetary trajectory design problems.
The 9th edition (GTOC9), named "The Kessler Run" is a competition that took place in 2017. The theme of the competition was active debris removal due to the growing problem of space debris. The objective of the competition was to find the best way to reach 123 debris in LEO in order to install on them deorbit kits after a 5 days rendezvous manoeuvre for every debris. The mission is set in 2060 in a hypothetical situation where space debris have become such a problem that LEO missions market size have gone beyond the trillion Euro.
In order to find the best possible solution, participants had to minimize the cost function

$$J = \sum_{i=1}^{n} C_i = \sum_{i=1}^{n} \left[c_i + \alpha \left(m_{0_i} - m_{dry} \right)^2 \right]$$

where C_i is the total cost of a submission and is composed by a term c_i that increases during mission timeline, and $\alpha \left(m_{0_i} - m_{dry} \right)^2$ is a term depending on the spacecraft mass: m_{0_i} is the spacecraft mass at the begining of the i-th mission and m_{dry} is the dry mass of the spacecraft. α is set to $2.0 \cdot 10^{-6} [MEUR/kg^2]$ and n is the number of submission necessary to reach every debris. The term c_i is computed as

$$c_i = \frac{t_{submission} - t_{end}}{t_{start} - t_{end}} (c_M - c_m)$$

where $t_{submission}$ is the start epoch of the i-th submission t_{start} is the starting epoch of the whole mission and is set to $t_{start} = 23467 [MJD2000]$, t_{end} is the ending epoch of the mission and is set to $t_{end} = 26419 [MJD2000]$ and c_m and c_M are, respectively 45 MEUR and 55 MEUR.

The competition states that spacecraft mass must be calculated with the Tsiolkovsky equation

$$m_f = m_i \exp\left(-\frac{\Delta V}{v_e}\right)$$

where $v_e = I_{sp} g_0$
All spacecraft have a dry mass of $M_{dry} = 2000kg$ and the maximum initial propellant mass is $m_p = 5000kg$, resulting in a maximum total starting mass of $m_{0_i max} = 7000kg$ for each submission. The operational constraints given are:

1. Between successive rendezvous manoeuvre there must be a maximum time of *30 days*. And, because the rendezvous manoeuvres take 5 days, *5 days* is the minimum time between successive rendezvous manoeuvres.
2. Between every submission there must be a minimum time of *30 days*
3. All mission events must happen between starting epoch $t_{start} = 23467[MJD2000]$ and ending epoch $t_{end} = 26419[MJD2000]$.
4. The osculating orbital periapsis r_p cannot be smaller than $6600000m$.

In the table are shown all the values set for the competition

	Value	Units
α	$2.0 \cdot 10^{-6}$	$MEUR/Kg2$
c_m	45	$MEUR$
c_M	55	$MEUR$
t_w	5	$days$
m_{de}	30	Kg
m_{dry}	2000	Kg
m_p	5000	Kg
r_{p_m}	6600000	m
μ	$398600.4418 \cdot 10^9$	$m3/sec^2$
J_2	$1.08262668 \cdot 10^{-3}$	$-$
req	6378137	m
I_{sp}	340	sec
g_0	9.80665	m/sec^2
Day	86400	sec
$Year$	365.25	$days$

Table 3.1. GTOC9 Values

The list of debris is given in a `.csv` file, for each debris a list of orbital parameters is given, all referred to a specific epoch, different for every debris. The parameters given are:

- *Reference epoch*

- *Major semiaxis,* **a**
- *Eccentricity,* **e**
- *Orbit inclination,* **i**
- *Right Ascension of Ascending Node*, **Ω**
- *Argument of periapsis*, ω
- *Mean anomaly,* **M**.

3.2 Starting point

The goal of the thesis was to optimize asolution to the GTOC9 competition that used graph theory in order to obtain better solutions. In this section will be briefly described the method and which points have been modified and optimized.

3.2.1 Graph theory sequence generation

The 8 years time lineset by the competition has been divided into 300 timesteps of 9.8729 days each.
It has been created an oriented graph in which transfers under a threshold of **370 m/s** (number that will be discussed in the next section) between debris in every timestep have been saved. The transfers taken into account are the ones for which the duration is 5, 15 or 25 days. These values, added to the 5 days necessary for rendezvous, resulted in total transfer times of 10,20 and 30 days.
Every node in the graph is defined by 2 parameters: the *debris ID*, that is a number between 1 and 123 and the *temporal frame* in which it finds itself, that is a number between 1 and 300.
Two graphs have been created: for both, as mentioned, the nodes represent the debris in all timesteps, while the edges represent the transfers: In the "g" graph the weight of each edge is given by number 1, 2 or 3 depending on the length of the transfer (respectively 10, 20 or 30 days) while on the graph "DV" the weight is given by the actual transfer cost in terms of ΔV.
A detail of "g" graph is reported in the following figure.

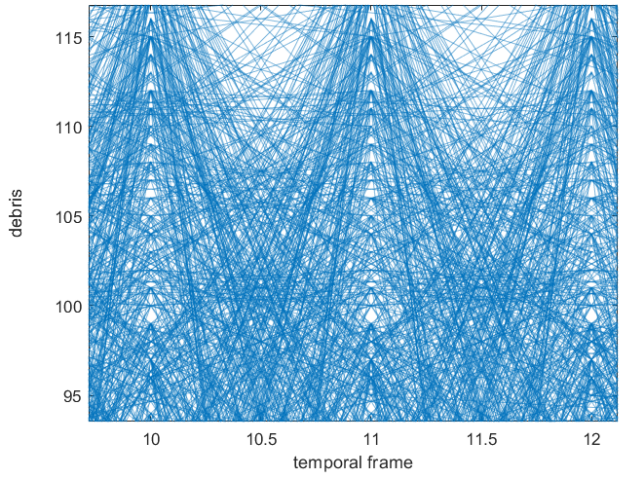

Figure 3.1. "g" graph detail

Once the graphs have been created, the sequence-generating algorithm starts: the sequences are created by the generation of 20 routes through the graph. Every route is independent from others and will create 20 different sequences from which only the best one will actually be used as the starting point from the final solution. The algorithm makes its way into the graph following the constrains given by the GTOC9. Whenever a route has no more possible continuation, it interrupts and the sequence continues, starting from another route that must not repeat reached debris.
The parameters to start the algorithm are: the maximum number of iterations for every route before the research interrupts and the maximum number of reached debris for each sequence before the algorithm interrupts. Those values are set to, respectively 500 and 105, but, as the ΔV parameter, those values will be discussed in the next section.
Every interruption of the research process will be made to coincide with the end of a submission and the beginning of the next one.
After the conclusion of this phase, every submission is checked if the constraint on the submission mass is respected. Since all the submissions will be optimized at the end of the whole process (including the one discussed in this paper) the limit on the submission weight is set at 8000 kg. If one submission exceeds this value, it gets splitted in 2 separate ones at the 30 days transfer closest to the middle of the submission. It is important to note that once the sequences are completed, not all the debris are placed and the solutions need additional processing in order to reach all 123 debris.

Once the starting sequences are ready, all useful mission data is saved in a

structure that will be continuously updated during the next phases. The main parameters saved in the structure, that will be used to process the solution in the next phases are:

- Length of every submission
- Reached debris order, position in time and subdivision in submissions
- The cost of every transfer between debris, in terms of ΔV
- The cost of every submission in terms of MEUR
- The cost of every submission in terms of mass
- The total cost of the mission at the point of the processing
- The duration of each time transfer, including the 5 days duration of the rendezvous, saved in a separate array.

In figure 1.6 is reported an example of the struct containing mission data.

```
                n:   [13 10 13 12 11 10 8 6 13 6 5 6 5 5]
             list:   [1×123 uint8]
              pat:   {1×14 cell }
            feas:   [14×1 double]
           estdV:   {1×14 cell}
      total_cost:   [1×14 double]
costo_medio_min:   [1×14 double]
            estM:   [1×14 double]
         massMEUR:   [1×14 double]
         timeMEUR:   [1×14 double]
            subm:   [14×13 uint8]
              TC:   831.5149
          patTot:   [1×123 uint8]
            left:   [1×0 double]
        timeline:   [1×123 double]
        interval:   [1×123 double]
```

Figure 3.2. Mission struct example

From this point on, the methods developed during this thesis work were applied, but for completeness, the methods developed in the previous solution are briefly described. The first method inserts, where possible, respecting GTOC9 mass and transfers lengths constraints, left debris at the beginning of the submissions, before the first debris. The second method creates new additional submissions at the end of the mission or in a pause long enough between submissions. The new submission

is created combinatorially inserting new debris in blocks of 7 debris, finding the best permutation in order to obtain the cheapest new submission possible.

Once all the debris are inserted, a process of overall optimization is applied to the solution, in order to find a new time disposition of the debris that could ensure at least a local minimum in the mission cost. This has been implemented by using the `fmincon` function in `MATLAB`®. Given a function to minimize and boundary condition in which the optimization must be found, `fmincon` is able to find the local minimum of the given function.
Since the variable to minimize is the total cost of the mission and the free variables are the transfer duration of every transfer and the time gaps between the submissions, the lower and upper bounds are arrays with the same length as the time array that contain, for each transfer or pause between submission, the lowest and highest possible value.

- The **lower bound** is an array with 123 elements, built as: `0 [5 ... 5] 30 [5 ...5] 30 ... 30 [5...5]`, where every sub-array of 5s represent the lower value for transfers in-submission and each one is as long as the submission it's referred to, while the 30s represent the minimum pause between submissions.

- The **upper bound** is an array with 123 elements, built as: `Δd [30 ... 30] Δd [30 ... 30] Δd ... Δd [30 ... 30]`. Every sub-array of 30s represents the maximum length possible for transfers in-submission and are as long as the submission they are referred to, while the Δds are the maximum value for pause between submissions and are set as the longest pause present in the solution that is going to be optimized with an addition of 1 days.

`fmincon` also needs two additional values **A** and **B**, that are the terms of the inequality $A \cdot x \leq B$ meaning that **A** is a 123 long array of ones, while **B** is the time difference between the starting and the ending epochs of the mission set by the GTOC9 in [*days*]. In the following figure is shown an example of the `fmincon` output of the overall optimization.

Another optimization is then carried on, trying to find possible swaps between placed debris in the submission using random permutation of a random number of debris between 2 and 6. After this and additional overall optimization using `fmincon` is performed since after some submissions may now contain new list of debris, the solution might not be still a local minimum.

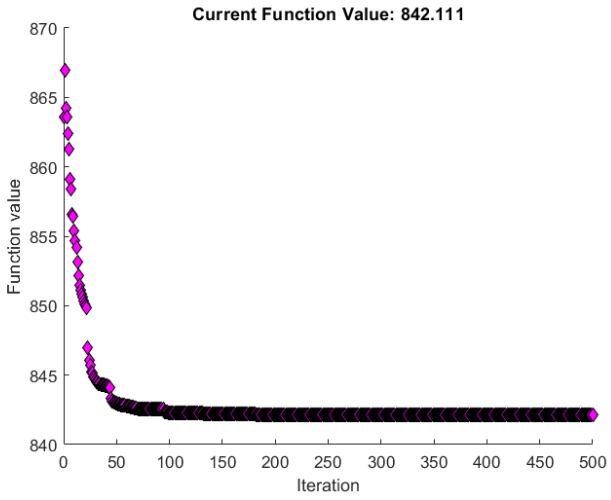

Figure 3.3. Example of `fmincon` overall optimization

Submission mass evaluation

Since for every rendezvous a deorbit kit is released, the total mass of ever submission cannot be calculated at the beginning of the submission: the initial mass of every transfer won't be equal to the final mass of the previous transfer. The method, therefore, starts at the end of each submission.
In every submission there will be $n-1$ velocity changes: $\Delta v_1, \Delta v_2, \Delta v_3, ...\Delta v_{n-1}$. For every transfer the initial mass is calculated via the Tsiolkovsky equation

$$m_i = m_f e^{\frac{\Delta v}{c}}$$

where $c = I_{sp} g_0$, with $I_{sp} = 340s$ and $g_0 = 9.8067$, as stated by the GTOC9. The mass right after the last rendezvous would be

$$m_{f,n-1} = m_{dry} + m_{de}$$

where m_{dry} is the dry mass of the chaser and is 2000 kg as stated by the GTOC9, while m_{de} is mass of the deorbit kit, set at 30kg. By applying the Tsiolkovsky equation for the $n-1$ transfer, the mass before the transfer is

$$m_{i,n-1} = m_{fn-1}$$

. The next iteration wuold be the same, with the exception that

$$m_{f,n-2} = m_{i,n-1} + m_{de} \neq m_{i,n-1}$$

The process is applied $n - 1$ times.

Chapter 4

Methods description

As mentioned before, from the point where only starting sequences are found, the objective was to develop one or more advanced methods to fill those sequences with the left debris in order to obtain a complete and competitive solution to the GTOC9 competition.
This goal has been achieved via the development of three main insertion methods, some of which are divided in smaller sub-methods:

- **Direct debris insertion method**, divided in:
  - 30 days gap
  - 20 days gap
- **Replacement - insertion method**
- **Generated - spot insertion method**, divided in:
  - End of the submission
  - In - submission

The last section of this chapter contains the description of the editing that has been made on the first section of the code that generates the starting sequences.

4.1 Direct debris insertion method

This method, as the title suggests, directly inserts left debris inside the sequences resulting from the first section of the code. To better explain how it works, the description will focus on the 30 days gap sub-method since the 20 days gap one uses the same functions and algorithms with different starting parameters.

4.1.1 30 days gap

The idea behind this method was to find 30 days transfers inside the already present sequences in order to fit another debris and, hence, two other transfers of smaller duration: one transfer from the debris before the 30 days transfer and the inserted one and another from this last debris to the one where the 30 days transfer ends.
As a first approximation the two shorter transfers were of 15 days each, so that their sum was exactly 30 days. But, since none of the transfers was exactly 30 days long (they were, in fact, 29.6187) as described in the previous chapter, it has been necessary to set the first transfer length to 15 days as before, and the second one to the time left for it from the gap. In this particular case of 30 days gap method the two transfers will last respectively 15 and 14.6187 days.
Of course, since transfers happened in certain times during the mission and between certain debris, not all the left debris were useful for every insertion: only certain combinations could be used, depending on transfers cost. It has been therefore necessary to calculate all the possible combinations in order to sort them and find only the useful ones. This process has been carried out via the implementation of a function called `mat_inserimento`.

`mat_inserimento` function This function has, as outputs, three main matrices called `prima_trasf`, `seconda_trasf` and `trasf_tot`, that contain all possible combinations of insertion of left debris in all the 30 days transfers present in all sequences and, for each combination, the cost of the transfer in terms of ΔV. In particular, `prima_trasf` and `seconda_trasf` contain, respectively, the cost of the first and of the second transfer, while in `trasf_tot` the sum of the two.

Since this function had to be reused also in the other methods, it has been written making it as adaptable as possible: putting as input, in addition to time array, debris orbital parameters and general mission data (such as list of sequences, left debris, etc.), also the time gap the user is interested in finding inside the sequences and the duration of the first transfer of insertion. Second transfer duration is not necessary as input since it depends on the gap and the first transfer.

All terms of `prima_trasf` and `seconda_trasf` have then been sorted, following the order from the cheapest to the most expensive in term of ΔV cost, excluding all transfers that would cost more than 2000 m/s because it would mean using calculating capacity of the computer for useless purpose as more than 2000 m/s for a single transfer would make the whole submission unusable. In order to fit a larger number of debris, as some insertion might present cheap first transfers and expensive second transfers or vice versa, the `fmincon` function has been used to optimize the time duration of the two new transfers generated after the insertion. Fixed the sum of the two transfers as the length of the 30 days gap transfer and the minimum and maximum values of the two transfers at 5 and 30 days (imposed by

GTOC9 constraints), `fmincon` returns the optimum duration of the first transfer, and hence the second. This result was generated finding the minimum of a new function `costins` written ad hoc for this optimization. `costins` is a simple function that locates the position of the 2 or 3 debris given as input and calculates the cost of those 2 transfers based on the duration of the first transfer, also given as an input. In particular, the duration of the first transfer is the free variable in the `fmincon` function, while the cost is the variable to minimize. `costins` is another function that would be reused in other methods, and is therefore capable of working with just 2 debris and one transfer, as it will be discussed in the next methods.

A new matrix has then been created called `mat_ins` that contains all useful data for insertion.
An example of one matrix written during the final solution is shown in table 4.1.

Total ΔV optimized $[m/s]$	1st transfer dur. optimized $[s]$	Left debris position	Gap position	1st transfer ΔV optimized $[m/s]$	2nd transfer ΔV optimized $[m/s]$
1049.89	10.96	26	11	679.65	370.25
1095.31	5.00	2	10	709.61	385.70
1162.52	10.79	12	9	755.89	406.63
1544.78	5.00	4	3	720.36	824.42
1623.40	5.00	27	5	986.04	637.36
2075.02	5.00	17	6	1059.66	1015.37
3408.40	14.75	8	14	1774.68	1633.72

Table 4.1. `mat_ins` example.

IDs in 3rd and 4th columns are to be intended as x and y positions in `trasf_tot` matrix, as in the code they resulted a lot more useful than the actual debris IDs. The last step was to try to insert the debris in the submissions, verifying at every loop that a maximum value of submission mass (8000 kg as final optimization will reduce this value) was not exceeded.
If the result was positive the code proceeded to actually insert the debris in the submission, and to give a string as output with necessary information:

```
Adding debris _ in submission _ at position _
The submission will weigh _ kg
```

Otherwise, the user would be informed via the output:

```
Too heavy
```

To better understand the *direct insertion method* a visual representation is shown

in Figure 4.1.

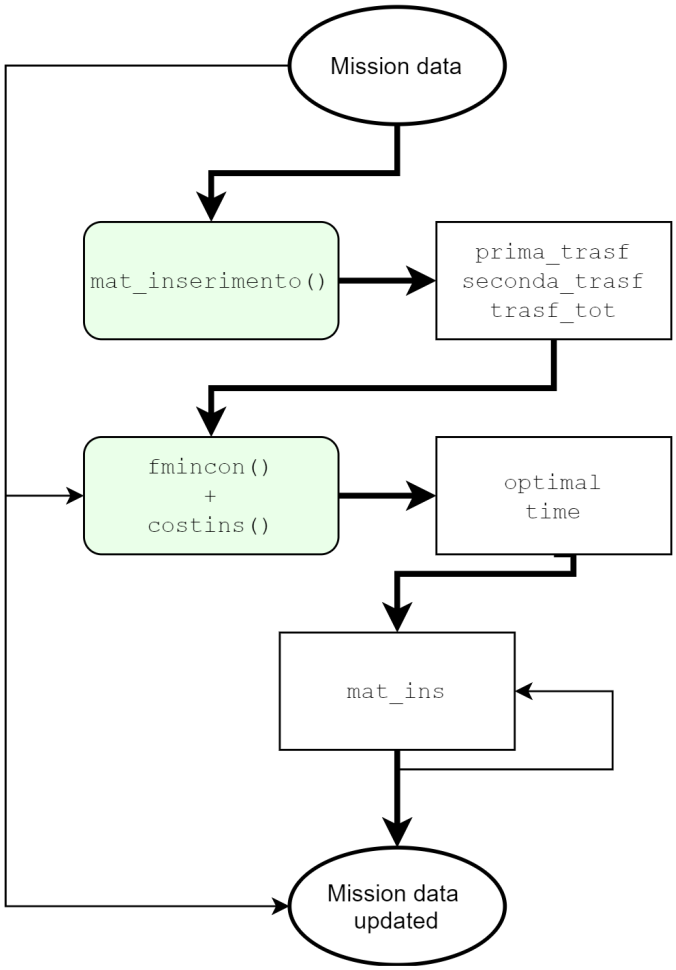

Figure 4.1. Visual representation of *Direct insertion* method

4.1.2 20 days gap

As mentioned before, the 20 days gap insertion method works the same way as the 30 days one, except for the inputs in the function `mat_inserimento`: the first transfer duration would be 10 days and not 15, and the gap to find in the submission is between 15 and 25 days. Exactly 20 days were not possible as input because of two reasons: the first is that gaps are almost never exactly 20 days, but rather 19.7458 for the same reason discussed in the previous paragraph, and also because, after the optimization in the 30 days gap method, many different values from 15 to 25 days may appear as duration of one of the two transfers.

4.2 Replacement - insertion method

This second method, as its name suggests, tries to find the best combination of replacement of a left debris with an already placed one, along with the placement of the new left debris.

Replacement

In order to reach this goal, the first step has been to find and sort in descending order of ΔV cost all the possible replacements. This has been done via the function `sost_cost`.

`sost_cost` function The function generates 5 matrices, each one with a specific purpose: `costo_prima`, `costo_dopo` and `guadagno` all have same dimensions (N left $\times$ N placed) and, respectively, contain the total cost of a debris (intendend as sum of tansfer before and after that debris) before the replacement, after the replacement and the difference between before and after the replacement. Other 2 matrices are generated that are simply the cost of the transfer before and the one after the replaced debris. It is important to note that the matrix `guadagno` might contain negative values, that means that the replacement in itself is not convenient, but may be a good solution if paired with an efficient insertion

Another new matrix has then been created, called `mat_sost` with all the sorted replacements that, overall, didn't exceed a total gain/loss of -2000. In this case 2000 has been considered a reasonable value to stop at since it allows the code to work with a greater number of combinations without slowing the process too much. In table 4.2 is shown an example of the replacement matrix `mat_sost`.

Left debris	To replace debris	Total gain $[m/s]$	Transfer to debris gain $[m/s]$	Transfer from debris gain $[m/s]$	Total cost $[m/s]$
79	26	287.16	287.16	0*	97.28
57	112	117.96	0*	117.96	526.57
79	61	-89.17	-693.39	604.22	967.24
79	6	-137.21	-79.13	-58.08	1234.43
9	64	-465.44	-465.44	0*	563.62
57	43	-623.33	-380.03	-243.31	1566.60
15	94	-684.67	0*	-684.67	852.98
57	54	-762.13	-516.10	-246.03	1568.18
57	59	-1096.27	516.62	-1612.90	2633.94

Table 4.2. `mat_sost` example

The figure represents the first 9 rows of the matrix generated in the final solution. The first two columns contain the IDs of respectively the debris that would replace and the debris that would be replaced. The 3rd column is the sum of the 4th and the 5th that shows the gain in transfer to and from replaced debris, while, as described in the figure, the last column is the total cost of the replaced debris (as sum of transfer to and from it).
* Null values in 4th and 5th columns appear in case a replacement happens at the end or at the beginning of a submission, where the transfer before or after the debris does not exist.

Insertion

Once this point has been reached, the insertion phase began. For every replacement a new left debris had to be placed. The procedure applied to find a gap where to place the new debris is similar to the *direct debris insertion* method: for every new left debris the function `mat_inserimento` has been applied singularly (so that matrices `prima_trasf`, `prima_trasf` and `trasf_tot` were only one row long) two different times, one with input parameters for a 20 days gap and the other with input parameters for 30 days gap. All the involved values, as well as the replacement values for that particular debris, have been saved in a new matrix called `plac_sost`.

`plac_sost` The first version of this matrix contains in every row the same info contained in the rows of `mat_sost` but in adddition it also contains all values useful for the insertion of the replaced debris (the values resulting from `mat_inserimento` for both 20 and 30 days gap insertions). Not all rows of `mat_sost` are present in `plac_sost` because if one insertion presents values of total ΔV that exceeds 1000 m/s for both 20 and 30 days transfers, the row gets deleted. A whole paragraph is dedicated to this specific matrix because it's the most important matrix of the method and determines its structure: the whole optimization and actual insertion part of the method is looped about its length. This is because once a certain combination is applied to the mission, all the other values in the matrix must be updated to the new mission data (new time array and presence of new debris that may appear after a replacement or after an insertion), and it is not possible to know a priori whether or not a certain combination can be applied to the solution.
The first version of `plac_sost` was therefore a guideline for the continuation of the method and would be updated at every iteration with positive result. In fact if a replacement between a left debris and two or more placed debris appeared two or more times in the matrix, only the first useful combination would be used (that should also be the less expensive) and the other would be ignored.

Optimization

To make this method more effective, every insertion and every replacement have been optimized using the function `fmincon`.
As in the *direct insertion* method, `fmincon` optimizes times using the function `costins` with as inputs 3 debris in the insertion optimization and in the replacement if it happens inside the submission and with 2 debris as inputs only for the replacement optimization if it happens at the beginning or at the end of the submission. This is because inside the submission, for both replacement and insertion, 3 debris are involved: the inserted/replaced one, the one before and the one after, and hence 2 adjacent transfers needed to be optimized, the same way as in the direct insertion method, while for the beginning or the end of the submission the involved debris were only the replaced one and the one after or before it, with just one transfer.

Replacement and insertion attempt

All optimal time values and, hence, updated ΔV values of insertions and replacements have been written in `plac_sost` matrix.
Once the row has been completed with all the values (divided in 20 columns), the minimum value between the ones on the 20 days gap insertion and the 30 days one has been chosen and with it all relative other values (position relative to that transfer, optimal insertion time) and replacement/insertion was attempted. If the result was positive the code gave, as output, the string:

```
Adding debris _ in submission _ at position _
The submission will weigh _ kg
Replacement happened in submission _ at position _.
Debris _ has been replaced with debris _
```

The result could be negative for three reasons:

- The submission in which there should happen would be too heavy
- The submission in which the insertion would happen would be too heavy
- The submission in which replacement and insertion would happen is the same and would be too heavy

For those three cases, the code gave three different outputs:

```
Replacement too heavy
Insertion too heavy
```

`Insertion and replacement too heavy`

It follows a visual representation of *replacement - insertion* method to better understand the logical flow.

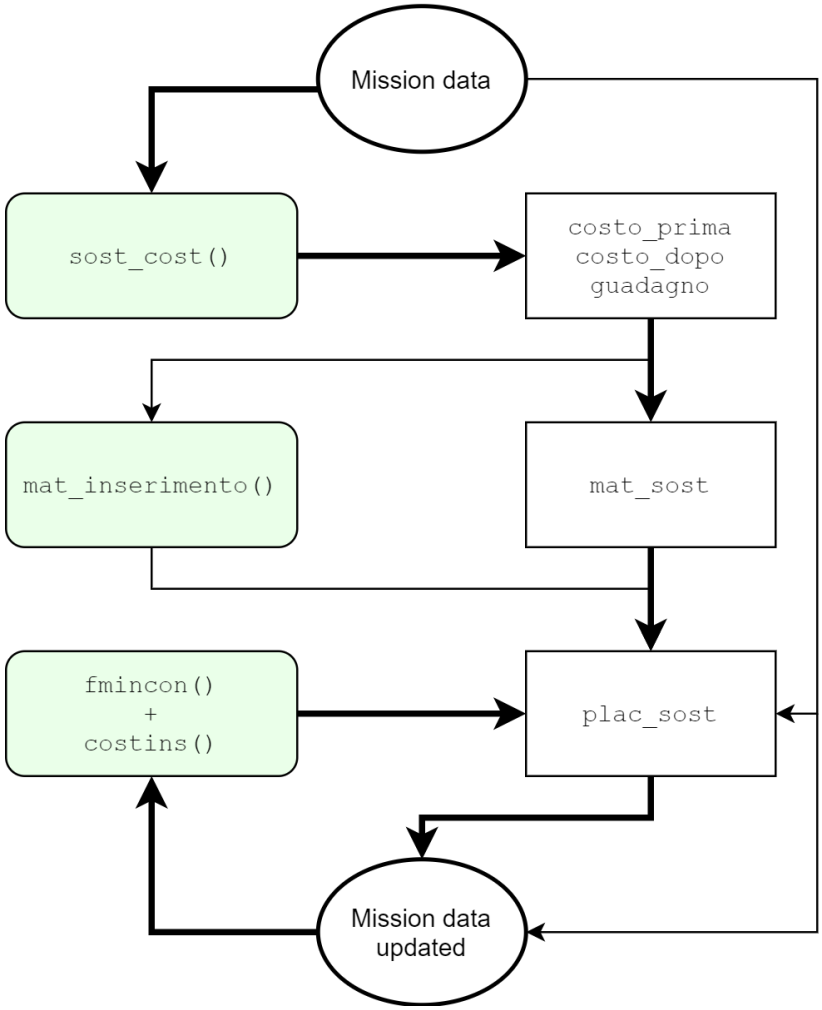

Figure 4.2. Visual representation of *replacement - insertion* method

4.3 Generated - spot insertion method

This third and last method tries to create new time gaps between debris in the submissions in order to insert new ones. To do that, it translates in time each debris in the target submission, respecting all GTOC9 requirements. The method is divided in 2 sub - methods: one that only deals with in-submissions insertion and the other that only deals with insertions at the end of the submission. This division has been made since, as it will be described, the two methods have similar logical processes, but their implementation follows two different strategies.

4.3.1 In - submission method

In order to make a clearer explanation, the method description has been divided in two steps: *optimal combination finding* and *spot generation and insertion.*

Optimal combination finding

Since in this method, differently from the first one where the gaps already existed, the gaps must be generated, there were no constraints in the position of the insertion and every transfer could be a potential insertion spot. It has therefore been necessary to calculate the cost of transfers between all placed debris and all left debris. Those transfer have been calculated with the duration of 10, 20 and 30 days in order to find possible short cheap transfers.
Once all the combinations have been found, they have been sorted in ascending order of ΔV cost and then divided in N_{left} different matrices in a cell, obtaining a list of ascending cost combinations for each left debris, the cell is called `leftgeninfo`.
Since each potential insertion has been calculated ignoring all the others, it was not possible to use just the cheapest combination for all debris: some may find themselves in the same position as others, so it has been necessary to find the best combination that ensured the potential insertion of all left debris without overlay of debris.
To obtain this result it was necessary to cycle at least the first 5 potential insertion of all left debris with all the other first 5 of the other left debris in order to obtain all possible combinations and, only then, select the less expensive that would respect the constraint of non - overlaying between left debris. This process has been done via the utilization of a recursive function since N nested `for` loops, where N is the number of left debris, were necessary, and it was not possible to modify the code based on a variable that changed for every different solution. The recursive function is called `opt_recursion`.

`opt_recursion` As mentioned before, this function has been used to calculate and save all possible combinations of the first five best potential insertions for each

left debris. The key characteristic in a recursion function is the fact that inside the function it calls itself, passing many parameters at the first level of the function, it is possible to interrupt the loop of recalling itself when needed. In this particular case, the function recalled itself as many times as there are left debris. Every time the limit depth is reached, the recursion goes back one step and increases one parameter, changing the combination with a different last insertion, once all the first five insertion of the last left debris have been saved, the recursion goes back two steps, increases the second-to-last parameter and repeats the process just explained. To better understand the process, in figure 4.3 is represented an example.

	1st level	2nd level	3rd level
1st iter	A1	B1	C1
2nd iter	"	"	C2
3rd iter	"	"	C3
4th iter	"	B2	C1
5th iter	"	"	C2
...	...	...	...
Last iter	A3	B3	C3

Figure 4.3. `opt_recursion` example

In the example every left debris is represented with a letter, A, B or C, and the number associated with each letter represents the potential insertion (1 is the cheapest and 3 is the 3rd cheapest), as it's possible to see, the number of the last debris increases for every iteration, and once the maximum number (3 in the example) the left debris that comes before increases.
In the actual function the numbers taken were 5, number that ensures almost certainly that the best combination will be calculated, and in the final solution the number of left debris was 6.
All combinations have been saved in a global matrix called `mat_comb` with $5^{N_{left}}$ rows, one for every combination.
In order to keep the calculating time in acceptable limits, the recursive function is called only if, at the point of the code where the *generated - spot* method is called, the left debris are less or equal to 7, that means 78125 combinations.

Once the combination matrix has been calculated, it has been sorted on ascending order of cost, where the cost is the sum of every ΔV for each potential insertion. Then it has been possible to choose the less expensive that could respect the constraint of no overlaying. The result is a saved in a matrix called `mat_comb` and used to attempt insertions. The result from the final solution is reported in table 4.3.

Left ID	Insertion cost	Before left ID	Insertion length
30	994.53	74	30
57	211.94	43	30
77	861.73	36	30
87	1709.93	38	30
93	904.40	79	30
102	1694.29	35	30

Table 4.3. `mat_comb`

Spot generation and insertion

The space necessary for the debris to be inserted in the target submission is the fourth column of every plus an additional 20 days to get from the placed debris to the one after it (20 days has been chose after several attempts, and can be considered a reasonable trade-off value). In order to create such space, an iterating process has been carried on, shortening by 1 day every transfer in the submission until the space is enough. The process shortens all necessary transfers following this method.

Spot generating algorithm Set the target transfer position in the submission as i, the shortening order follows the order

i-1	i+1	i-2	i+2	i-3	i+3

for every shortening, the target transfer gains 1 day and the process goes on until the target length is reached. If the position of the transfer to shorten exceeds the submission, the shortening starts again from the first transfer before or after the target transfer. A numerical example is shown in figure 4.4.

Position	0	1	2	3	4	5	6	7	8
Iter 1	0	0	0	-1	+1	0	0	0	0
iter 2	0	0	0	-1	+2	-1	0	0	0
Iter 3	0	0	-1	-1	+3	-1	0	0	0
Iter 4	0	0	-1	-1	+4	-1	-1	0	0
Iter 5	0	-1	-1	-1	+5	-1	-1	0	0
Iter 6	0	-1	-1	-1	+6	-1	-1	-1	0
Iter 7	0	-1	-1	-2	+7	-1	-1	-1	0
Iter 8	0	-1	-1	-2	+8	-1	-1	-1	-1
Iter 9	0	-1	-2	-2	+9	-1	-1	-1	-1
Iter 10	0	-1	-2	-2	+10	-2	-1	-1	-1

Figure 4.4. Spot - generation algorithm example

The first transfer remains untouched because it represents the time between two submissions and it's not an actual transfer. Moreover, by subtracting and adding the same amount of time in every step doesn't modify the temporal position of the submission, leaving untouched all the other submissions. Some constraints, however, need to be applied in order to respect GOTC9 rules: all transfers must not be reduced under 5 days, if it happens the code skips to the next shortening, if all transfer reach 5 days length without having reached the desired length for the target transfer, then the transfer target is shortened by 5 days, and the process starts again from the beginning, this will be repeated until even the target transfer would rich 5 day or less. At that point the debris cannot be placed and the code skips to the next debris.

Once this point has been reached, a complete optimization of the submission has been carried on, in order to remove all possible unbalances due to the insertion. In this case the bounds to the optimization were that each transfer inside the submission had to be between 5 and 30 days, while the first and the last transfers remain untouched.
This kind of algorithm allows not to modify the overall structure of the submission, this feature is important since the submission optimization works with local minima: if the entire structure of the submission (that should be close to the optimum due to the sequence finding method and the other optimizations) is compromised, the result after the optimization might not be as good.
This time the mission is update with the modifications only if its total mass is equal or less than 7300 kg, because, even though a total optimization on the submission has been conducted,the overall optimization will reduce its mass even more. The code outputs, if the attempted insertion gives positive results, are:

```
Adding debris _ after _
The submission will weigh _ kg
```

Otherwise, if the submission exceeds 7300 kg:

```
Too heavy
```

and, if as mentioned before, it is impossible to insert the debris due to time limitations

```
Not enough space for insertion.
```

Here is shown a visual representation of the method.

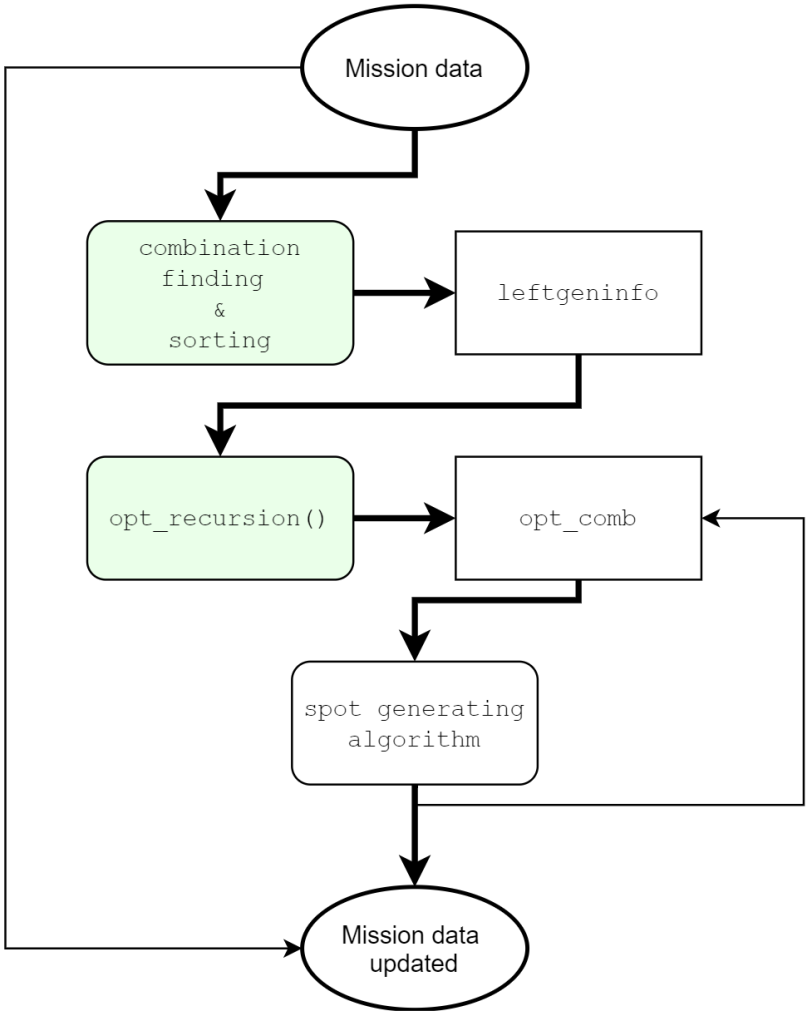

Figure 4.5. Visual representation of in-submission spot generation method

4.3.2 At the end of the submission method

The process followed in this sub - method is similar to the one described in the previous section, but, having less combinations to deal with, follows a different path to reach the insertion phase. The first step, has been to find, once again, all possible combinations between left and last debris in every submission. Since the probability of finding shorter but cheaper transfers is a lot lower than in the previous case, all the combinations have only been calculated with 10 days transfers, also because the final optimization would find the best transfer duration.
Again, all the combinations are sorted and the first five for each left debris are saved in a cell called `leftfinaliinfo` with all necessary insertion values: starting debris, left debris, transfer cost and transfer duration.

The insertion phase then starts, attempting the insertion of every possible combination, starting from the first of all left debris, once at a time. Every time an insertion is attempted, a vector called `ins_finale` is overwrote with the data of the insertion. An example of the vector `ins_finale` is reported in table 4.4.

The spot - generation follows the same algorithm of the previous sub-method, the

Left debris	Transfer cost $[m/s]$	Debris before insertion	Transfer duration
30	2271.3	32	10

Table 4.4. `ins_finale` example

only difference is that the shortening only happens backwards in the position, since the space to be generated is always at the end of the submission. In this case at least 30 days are left after the insertion to respect the 30 days between submissions constraints (the insertion could leave more than 30 days after the last debris if the gap is already greater than 40 days). Once the spot is generated and the debris is inserted in the submission, an optimization of the submission is again carried on with the same constraints as the previous sub-method.

For every iteration, if the result is positive, with the submission mass under 7300 kg, all future iterations will ignore the more expensive insertions for the inserted debris and keeps iterating the others. The code output is the same as the sub-method before. A visual representation of the sub-method is shown in figure 4.6.

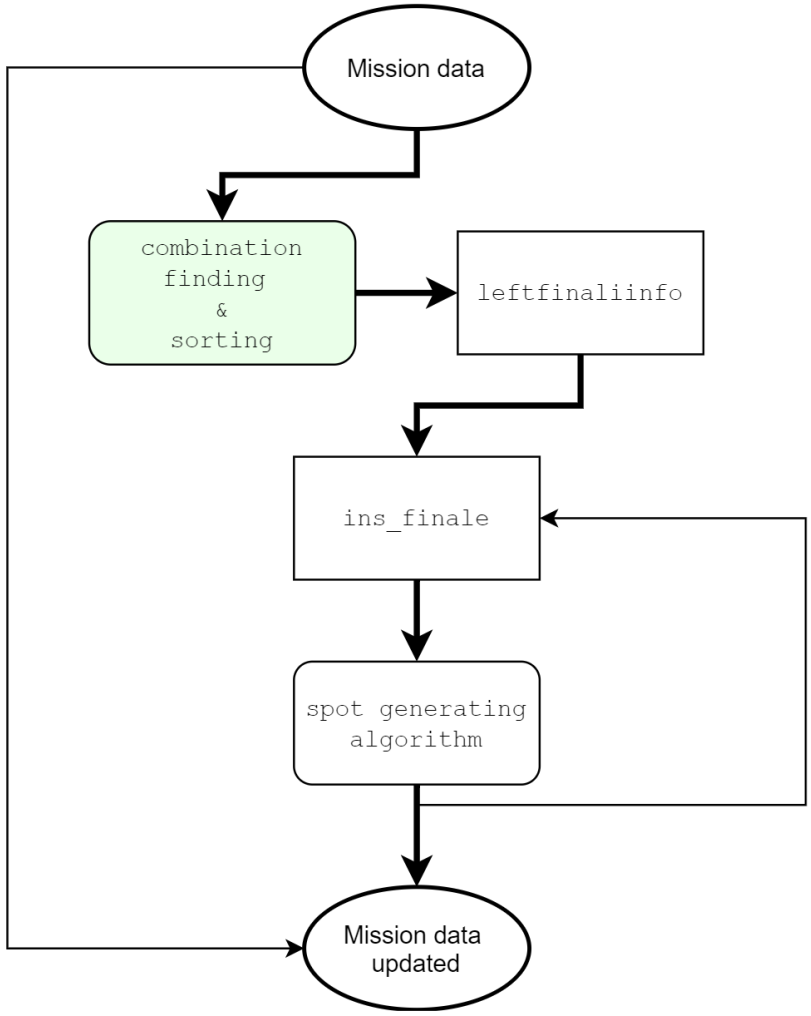

Figure 4.6. Visual representation of *at the end of the submission* spot generation method

4.4 Parameters editing

Based on the nature of the methods implemented during the work, the best feature a preliminary solution should have (where preliminary solution is the solution before the filling phase, just after the graph-method finding phase) is having short, cheap submissions. Surely, having shorter submissions means also having more submissions in order to have a reasonable number of already placed debris.
To obtain a result that could respect all those requirements, it has been necessary to modify some of the parameters responsible of the generation of the preliminary solution. The modified parameters are: ΔV *threshold for graph generation*, *number of iterations for sequences generation*,ΔV *weight on time dimension* and *debris*

number threshold to interrupt sequences generation.

ΔV threshold for graph generation

This value has been changed from an initial **370 m/s** to a lower **340 m/s**. This choice has been made because by reducing the maximum value of possible links between nodes, also the number of edges is reduced. This means that less interconnections are possible and therefore finding long submissions becomes harder: with less links there are less possible combinations from which is possible for the code to choose.

Number of iterations for sequences generation

From **500** iterations, the value has been modified to 100, so that the code has less time to analyse combinations and therefore find long submissions.

ΔV weight on time dimension

Before the implementation of the described modifications, the solution tended to have longer submissions, especially at the beginning of the mission, that means having less space to insert new debris. This particular modification has been specifically thought to solve this problem.
The first idea was to modify the weight of ΔVs used in the generation of the graph linearly along the development of the mission, meaning that the code would see the first transfers heavier than they actually were.
The problem with this kind of solution was that the ΔVs became too low already in the second or third sequence, meaning that the modification was effective only in the very first stages of the process.
The final solution has been achieved via an iteration process in order to find the best weight to give and from which point, this because the latest sequences, with a weight too high, resulted in submissions too short (1 or 2 debris). The solution sees a multiplication of calculated ΔVs for a factor that in the beginning of the mission has a lower decrease ratio than in the end. To do so, a cubic trend has been chosen, properly multiplied by factors obtained via an iterative process. The factor decreases until the 225th timestep has been reached. This choice has been made since the solution never occupies the entirety of the available time and over 225 timesteps it has been observed that the code would struggle too much to find useful links with a still increased ΔV weight. The factor follows the trend:

$$f = \frac{\sqrt[3]{i - 225}}{7.5} + 1 \qquad i = 1, 2, 3, ..., 300$$

The i represents the position of the ΔV in time dimension, since the time vector is an equispaced array of 300 terms between the starting moment of the mission

(from GTOC9 requirements) and the stopping moment. In figure 4.7 is shown the curve representing the trend of the factor *f*.

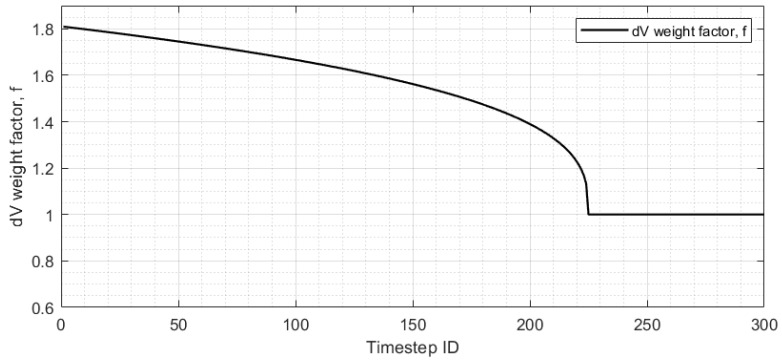

Figure 4.7. Visual representation of ΔV weight factor f

Debris number threshold to interrupt sequences generation

This value has been changed from **105** to **100**. This modification was made because even with the changes described in this section, the last submissions tended to be too short, passing from 105 to 100 stops the sequence finding in a way that all submissions present acceptable lengths with a reasonable amount of starting debris (the final solutions starts from 94 debris).

Chapter 5

Results

5.1 Results before debris insertion

Here are shown the results right after sequence finding and before the application of the methods discussed in the previous chapter. In the table are represented the list of debris for each submission. The total number of reached debris at this point was 94, that is less than the totality of debris, with 29 missing. It therefore needed additional processing.

Submission ID	Number of debris	Debris ID
1	11	16 115 121 39 104 96 58 17 119 51 24
2	8	31 108 62 86 20 48 42 72
3	9	44 116 3 46 83 8 27 71 64
4	11	53 29 74 65 4 70 67 13 91 11 32
5	9	23 14 33 60 99 18 100 84 47
6	5	112 59 92 61 26
7	7	82 120 105 52 109 38 25
8	5	94 10 78 95 101
9	9	68 41 63 2 37 90 1 36 123
10	4	80 111 98 66
11	4	88 117 21 28
12	5	89 114 106 19 5
13	4	69 34 81 107
14	3	49 7 97

Left debris	6 9 12 15 22 30 35 40 43 45 50 54 55 56 57 73 75 76 77 79 85 87 93 102 103 110 113 118 122

Table 5.1. Solution sequences before debris insertion

As it is shown in the table, the biggest submission contains 11 debris. This result is due to the edit of the parameters in the first section of the code also described in the previous chapter. The submissions are, in general, smaller compared to the results given without such modification. An example of results is now given: the table shows the mean values of 20 different sequences given by the code without the editing. All values given in the table in the non-edited row are to be considered as a mean of the 20 sequences.

	Max debris number	Min debris number	Number of subm	Mean subm length
No edit	14.9474	3.6842	10.2105	9.7248
Edit	11	3	14	6.7143

Table 5.2. Non-edited and edited starting sequence comparison

As mentioned before the edited sequences are, in general, shorter than the non-edited ones: 6.71 vs 9.72 and, proving that the editing has been successful, the edited solution presents a higher number of submissions, 14 vs 10.21. In the edited version the solution leaves more space for the insertion of left debris.

5.2 Results after debris insertion

The methods developed and implemented during the work were then applied.
In order to make the most of the used methods, the order in which they have been applied to the starting sequence is:

1. Direct debris insertion method, 20 days gap (*first application*).
2. Direct debris insertion method, 30 days gap.
3. Direct debris insertion method, 20 days gap (*second application*).
4. Replacement - insertion method (*first application*).
5. Generated - spot insertion method (*at the end of the submission*).
6. Generated - spot insertion method (*in - submission*).
7. Replacement - insertion method (*second application*).

The choice of the order was made following an iterative trial and error process but the validity of the result is easy to confirm: the first three sub-methods are applied firstly because it does not modify the temporal position of the debris that are not

directly involved in the placement (the debris before the one placed, the one placed and the debris after the one placed). The 20-30-20 sequence is the best one of the possible combinations due to the weight of the insertions: the 30 days placements weight less than the 20, so applying the 30 days placement before the 20 days one leaves less space for those insertions, reaching the maximum mass imposed for the submission with less insertion. The second application of the 20 days placement is necessary because new 10 to 20 days gaps have been created after the 30 days insertions.
The *replacement-insertion* method has been applied two times because, being highly sensible to debris mission changes, such as debris order or position in time, after the application of the *generated-spot* method, new insertion might be possible. For each method is shown how many debris were collected.

- *Direct debris insertion method, 20 days gap*: **7 debris**.
- *Direct debris insertion method, 30 days gap*: **10 debris**.
- *Direct debris insertion method, 20 days gap*: **2 debris**.
- *Replacement - insertion method*: **4 debris**.
- *Generated - spot method, in - submission*: **1 debris**.
- *Generated - spot method, at the end of the submission*: **5 debris**.
- *Replacement - insertion method*: **none**.

The absence of debris in the last insertion phase is due to the completion of the solution before this last phase, making it useless in this particular solution. A complete view of debris insertion and replacement given as the output from the code can be found in Appendix A - Code Output
The list of debris after the insertion phase is shown in the table below.

Submission ID	Number of debris	Debris ID
1	13	16 56 115 121 39 104 96 58 17 119 122 51 24
2	10	31 108 62 75 86 12 20 48 42 72
3	13	44 116 3 46 110 83 8 22 40 $\bar{64}$ 27 71 $\bar{9}$
4	12	53 73 29 74 65 4 70 67 13 91 11 32
5	11	23 14 33 60 99 18 100 84 47 87 102
6	10	112 54 43 57 59 50 92 $\bar{79}$ 6 26
7	8	82 120 105 52 $\bar{25}$ 109 38 $\bar{76}$
8	6	$\bar{15}$ 10 $\bar{94}$ 78 95 101
9	13	68 85 41 63 2 37 103 90 1 55 36 123 77
10	6	80 111 98 45 66 93
11	5	88 117 113 21 28
12	6	89 114 106 118 19 5
13	5	69 34 81 $\bar{61}$ 107
14	5	49 7 35 97 30

Table 5.3. Solution sequences after debris insertion

In the table every underlined ID represents a debris that has been added to the sequence, the IDs with the cap are the ones which have been replaced during the *replacement-insertion* phase and debris underlined and with cap are the ones placed after being replaced.
It is possible to notice that the number of placed debris is 29 (the number of left debris before the insertion) and the number of replaced debris is 4, that is the number of debris placed with the *replacement-insertion* method.
Here are shown two tables containing time steps and the ΔVs of transfers between every debris.

Subm ID	Δt [days]
1	5 24.6 29.6 9.9 9.9 19.7 29.6 9.9 29.6 24.6 5 9.9
2	59.2 9.9 9.9 5 24.6 5 14.7 29.6 19.7 9.9
3	30 9.9 29.6 9.9 5 14.7 29.6 5 5 14.7 4.9 9.9 30
4	39.1 5 14.7 9.9 29.6 9.9 9.9 9.9 29.6 9.9 9.9 9.9
5	59.2 12.7 30 5 5 30 30 5 5 30 25.3
6	68.85 5 5 5 30 5 14.0 5 5 5
7	59.2 9.9 9.9 9.9 24.6 9.9 29.6
8	68.7 29.6 5 24.6 9.9 29.6
9	59.2 17.7 30 29.3 28 30 5 20.4 5 5 17.0 5 5
10	108.5 5 5 5 14.4 30
11	78.9 9.9 5 24.6 9.9
12	118.5 9.9 9.9 24.6 5 9.9
13	148.1 9.9 19.7 5 14.7
14	177.7 30 30 30 30

Table 5.4. Time steps after insertion, before optimization

Subm. ID	ΔV [m/s]	Mass [kg]
1	236.1 304.2 128.5 164.7 91.4 132.3 184.2 170.1 183.7 1034.8 1168.5 87.4	7031.6
2	144.2 183.7 312.0 448.5 724.4 829.8 146.2 185.3 150.7	5584.6
3	99.9 179.6 139.2 984.2 669.6 183.2 251.6 187.7 572.5 66.3 157.2 409.6	7175.9
4	1061.5 1029.9 137.1 146.3 93.7 97.5 120.2 174.5 194.6 154.5 191.6	6266.6
5	141.4 120.5 191.4 71.8 190.0 229.6 226.4 220.9 1883.5 592.5	6854.7
6	644.6 161.5 400.8 692.6 332.5 498.7 534.3 95.0 580.1	7077.1
7	136.8 62.6 153.1 175.2 45.5 152.9 973.8	3584.4
8	853.7 291.3 351.5 180.9 187.8	3729.4
9	239.6 272.7 164.7 181.1 260.0 301.5 382.2 78.9 516.9 367.6 155.3 886.9	6896.7
10	160.0 198.1 632.7 712.6 2436.7	7181.8
11	226.5 701.2 597.7 162.4	3486.8
12	261.7 205.4 1216.0 1525.7 256.9	5935.1
13	83.1 280.5 451.1 117.3	2787.2
14	168.2 960.2 610.1 1684.5	5791.1

Table 5.5. ΔVs and submission masses after insertion, before optimization

Almost all values respect the constraints given by the competition, except for a 4.9 days present in submission number 3. This result does not mean that the solution is not consistent, it is, in fact, due to the approximation to 10, 20 and 30 days in the first part of the code: those values are, instead, a little smaller (by 1/10 of unit) and the insertion of exactly 15 and 10 days transfers lead to a 4.9. This value does not imply great differences in the result (having 5.0 changes theΔV by only 2 m/s) and it will eventually be increased to a value higher or equal to 5 during the final total time optimization using the *fmincon* function. As it's possible to see, already before optimization, all the masses never go over 7500 kg, value that would be easy to be reduced under 7000 kg during the final optimization. The speed changes are, in general, reasonably small, with 9 transfers over 1000 m/s and 29 over 500, but, as it will be shown, almost all these values will be reduced during the optimization.

At this point of the solution processing, the overall mass of the mission was 79383 kg and the total cost was 1140.9 Million €.

5.3 Final results

Since all the debris were placed, it was then possible to apply the optimization already implemented in the starting code, applied as follows:

1. Overall time optimization
2. Combinatorial optimization
3. Overall time optimization

A double *overall time optimization* has been necessary since the order of the debris in the sequences had changed and the local optimal time found during the first optimization might not have been the same. In fact, during this second optimization about an additional 3% in terms of total cost has been saved. Further iterations of this process were not possible since the *combinatorial optimization* reached saturation and it was no longer possible to find new useful permutations.
This process has been performed several times in order to find a pseudo-optimal solution since the *combinatorial optimization* makes use of randomization, meaning that each run gave a different solution. The final total mission cost was

831.5449 Million €.

In the two following tables are shown the final results of time-step and ΔVs.

Subm ID	Δt [days]
1	0 5 27.1 23.7 5 14.4 21 30 7.3 30 30 5 30
2	30 5 5 5 15.8 30 28 29.1 5 5
3	30 5 22.4 5.6 16.3 30 30 5 5 23.7 11.2 5 5
4	30 28 19.1 15.1 22.3 5 6.6 16.1 30 30 5 27.9
5	30 21 23.2 5 5 26.1 23.6 5.9 5.0 30 30
6	30 22 5 29.4 30 5 25.6 5 5.0 29.8
7	30 5 6.1 5 26.1 5 5 30
8	30 30 5 28.2 30 30
9	75.1 29.2 21.5 30 30 30 5 25.3 5 5 25.6 23.6 5
10	30 26.2 9.7 5 30 26.2
11	63 15.4 5 30 14.2
12	111.2 22.5 9.6 30 5 30
13	106 29.4 30 5.0 29.9
14	167.5 30 30 30 30

Table 5.6. Final time step solution

Subm ID	ΔV [m/s]	Mass [tns]
1	235.1 198.8 151.2 163.6 63.5 119.0 177.7 169.6 159.2 363.8 85.7 557.8	4680.4
2	110.0 149.0 298.6 369.3 173.7 88.8 163.9 135.6 168.0	3651.6
3	71.0 297.8 231.0 51.5 135.0 94.0 346.4 190.1 224.7 88.8 72.0 324.0	4294.9
4	590.0 213.8 127.0 151.3 84.2 98.5 107.9 178.7 117.9 71.0 207.2	4075.2
5	138.3 74.9 128.4 70.5 217.8 271.6 153.4 238.9 1713.6 481.9	6146.0
6	278.4 101.9 320.2 233.6 250.7 431.0 176.7 79.5 283.5	4213.2
7	127.7 99.2 113.0 168.7 44.4 152.0 962.3	3551.0
8	226.0 156.9 180.1 406.8 462.0	3260.3
9	210.5 101.0 161.6 172.0 273.0 259.5 376.6 62.9 365.0 365.7 180.4 170.1	5053.6
10	95.7 55.6 287.6 1751.8 239.7	4362.2
11	177.6 226.3 209.6 153.0	2656.2
12	142.1 204.9 267.6 242.0 496.2	3185.2
13	83.2 195.5 203.6 97.5	2513.7
14	166.0 306.0 563.7 1133.5	3997.7

Table 5.7. Final ΔVs and submission masses.

All values in the time step table respect the requirements specified in the GTOC9 competition: all time steps at the beginning of the submission are greater or equal to 30 days, except the first values that equals zero, meaning that the mission will start exactly at the starting moment of the time window in which the mission must be performed, while the moment at which the last debris is caught is on the day 26201 [MJD2000], 217.95 days before the deadline of the mission (26419), requirement of the competition. All the other values are between 5 (that is exactly the time necessary for the rendez-vous) and 30 days (the maximum time step for in-submission time steps). As mentioned before, overall, the ΔVs have drastically decreased, showing only 3 values over 1000 m/s and 7 values over 500 m/s.

Figure 5.1 shows that almost the entirety of the mission happens with time steps less or equal to 30 days, with the exception of the last third of the mission where peaks are distinguishable: they represent the time steps at the beginning of mission 9,12,13 and 14. This phenomenon is due to the fact that during the first sequence finding, the first submissions were the easiest to find, despite the editing applied in that section, therefore those first sequences are more likely next to the optimum solution and didn't need great time translation in order to get low speed changes.

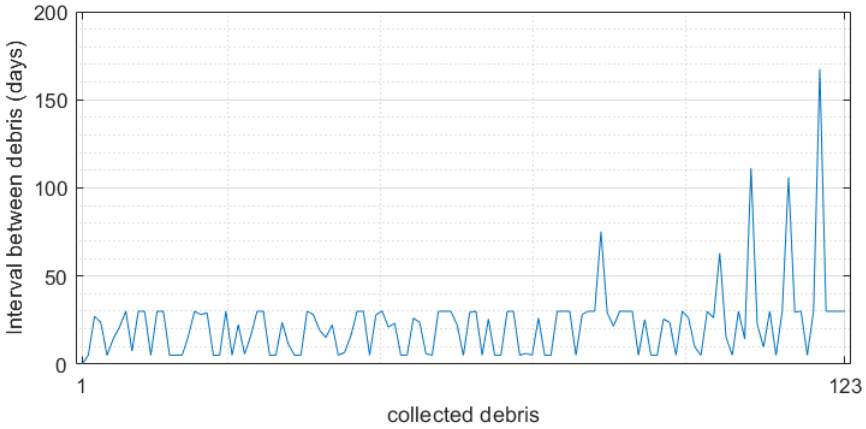

Figure 5.1. Mission time steps

5.4 Results comparison with starting solution

The first noticeable difference with the starting solution is the overall cost of the mission: this first solution has a cost of 865 Million €, whereas the new one costs 831.54 Million €, with a 3.87% gain. The most important values are shown in the table below.

	Old solution	New solution
Shortest subm. length	4	5
Longest subm. length	14	13
Mean ΔV change	285.46	246.94
Mean subm.mass	4285.1	3974.4
Subm. length std dev	3.19	3.26
Mean time-step	21.73	21.10

Table 5.8. Values comparison between old and new solution.

In general, it is possible to say that the results in the new solution are better: it can be seen from the mean ΔV changes, which show a gain of about 13.5% and the overall mean submission masses, with a gain of 7.25%. The other values in the table show that, even though the new solution has maximum and minimum values of submission length closer to the mean value of 8.79, the standard deviation for all lengths is smaller in the old solution, meaning that in general submissions lengths are closer to the mean value. Even the time steps are on average greater in the old solution with a theoretical smaller ΔV change. Despite those last two results, the new solutions give a lower cost, meaning that, in fact, it contains better debris pairing, proving the effectiveness of the developed insertion methods.

Here follows a graphic description of ΔVs trends for old and new solution. As can

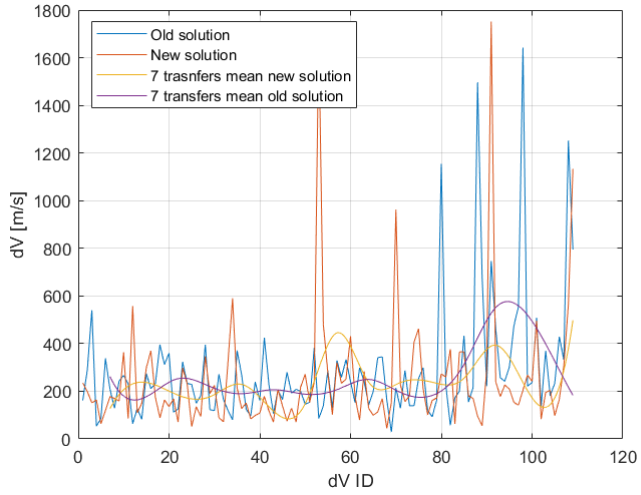

Figure 5.2. ΔVs trends of old and new solutions.

be seen in the graph, in the first half of the mission, the trend is almost equal for the two solutions, with the exception of one high peak. The main difference is found in the last part, where higher ΔVs value are concentrated for the old solution. This phenomenon is due to the filling method used in the old solution: the last submissions are less optimized than the others. The yellow and purple lines follow the mean value of ΔV every seven velocity changes: in the old solution is present an important increment in the last part of the mission, proving that those submissions are the heaviest one in terms of mean ΔVs.

Chapter 6

Conclusions

The goal of this work was trying to improve a solution that had a promising approach: the use of graph theory in order to solve a complex time-based problem.
The results obtained can be considered good and competitive if compared to the results published on the site of GTOC9, potentially placing this final solution in 5-th position.
The differentiation of the sequence filling process in three different methods has revealed to be a good strategy: it has made possible to exploit several features of the preliminary solution in different ways, such as the insertion of debris in the long gaps inside the submission, the substitution of debris in order to obtain additional gain and the pairing of two unpaired debris, where it was convenient, generating the optimal time gap.
Surely, confronted to the previous solution, the one developed in this work shows a general increase in homogeneity, and a decreased tendency to have better starting submissions. All these features resulted in an considerable gain in overall total cost. This is because, instead of creating new submissions in order to fit new debris in the mission, the methods modify already existing ones. This particular characteristic, however, does not ensure a complete solution for every starting sequence unlike the previous method: in order to reach all 123 debris it has been necessary to run the code multiple times and try different starting sequences.
It's important to note that such solution doesn't imply great efforts in terms of computational time, that is the key of the reason behind the choice of the graph theory application.

Each single method played an important role in the development of the solution even though some have demonstrated to be more efficient than others, leaving space for additional improvements:

- Improve the *spot generation* method in order to run the whole method at once for every step in order to obtain solution that refers to the update solution every iteration.

- Modify the overall optimization at the end of the solution in order to make the best use of the time available for the mission given by the GTOC9.
- Generalise the three methods and turn them into a single one, capable of choosing the best solution for every debris, avoiding to leave less debris for the last methods applied.

Appendix A

Code output

Here is shown the code output of the insertion phase only, divided by the method or the sub-method applied.

Direct debris insertion method, 20 days gap

```
Adding debris 103 in submission 9 at position 6
The submission will weigh 4.489249e+03 kg
Adding debris 6 in submission 6 at position 5
The submission will weigh 3.709252e+03 kg
Adding debris 50 in submission 6 at position 3
The submission will weigh 4.693769e+03 kg
Adding debris 12 in submission 2 at position 5
The submission will weigh 4.666559e+03 kg
Adding debris 110 in submission 3 at position 5
The submission will weigh 4.834577e+03 kg
Adding debris 73 in submission 4 at position 2
The submission will weigh 6.216988e+03 kg
Adding debris 35 in submission 14 at position 3
The submission will weigh 6.652574e+03 kg
```

Direct debris insertion method, 30 days gap

```
Adding debris 22 in submission 3 at position 8
The submission will weigh 5.453460e+03 kg
Adding debris 56 in submission 1 at position 2
The submission will weigh 3.975039e+03 kg
Adding debris 85 in submission 9 at position 2
The submission will weigh 5.204420e+03 kg
Adding debris 75 in submission 2 at position 4
The submission will weigh 4.812312e+03 kg
```

```
Adding debris 55 in submission 9 at position 10
The submission will weigh 6.616972e+03 kg
Adding debris 54 in submission 6 at position 2
The submission will weigh 5.399492e+03 kg
Adding debris 113 in submission 11 at position 3
The submission will weigh 3.504305e+03 kg
Adding debris 45 in submission 10 at position 4
The submission will weigh 3.802879e+03 kg
Adding debris 122 in submission 1 at position 11
The submission will weigh 7.094140e+03 kg
Adding debris 118 in submission 12 at position 4
The submission will weigh 5.991400e+03 kg
Too heavy
```

Direct debris insertion method, 20 days gap

```
Adding debris 40 in submission 3 at position 9
The submission will weigh 6.214456e+03 kg
Adding debris 43 in submission 6 at position 3
The submission will weigh 6.763903e+03 kg
Too heavy
```

Replacement - insertion method

```
Adding debris 61 in submission 13 at position 4
The submission will weigh 2.780578e+03 kg
Replacement happened in submission 6 at position 7.
Debris 61 has been replaced with debris 79

Adding debris 64 in submission 3 at position 10
The submission will weigh 7.198048e+03 kg
Replacement happened in submission 3 at position 12.
Debris 64 has been replaced with debris 9

Adding debris 94 in submission 8 at position 3
The submission will weigh 3.037593e+03 kg
Replacement happened in submission 8 at position 1.
Debris 94 has been replaced with debris 15

Insertion too heavy
Adding debris 25 in submission 7 at position 5
The submission will weigh 3.554627e+03 kg
```

```
Replacement happened in submission 7 at position 7.
Debris 25 has been replaced with debris 76

Replacement too heavy
Insertion and replacement too heavy
Insertion and replacement too heavy
Insertion and replacement too heavy
```

Generated - spot method, in - submission

```
Too heavy
Adding debris 57 after 43
The submission will weigh 7.120177e+03 kg
Too heavy
Too heavy
Too heavy
Not enough space for insertion
```

Generated - spot method, at the end of the submission

```
Too heavy
Adding debris 77 after 123
The submission will weigh 6.947280e+03 kg
Adding debris 87 after 47
The submission will weigh 5.373065e+03 kg
Too heavy
Adding debris 102 after 87
The submission will weigh 6.989545e+03 kg
Adding debris 30 after 97
The submission will weigh 5.912012e+03 kg
Adding debris 93 after 66
The submission will weigh 7.110361e+03 kg
```

Bibliography

[1] *Space debris by the numbers*, the European Space Agency, updated on 9 November 2021.

[2] *Guide to space debris*, spaceacademy.net.au, 26 August 2019.

[3] Selena Mastrodonato, *Rimozione di debris multipli in LEO: applicazione della teoria dei grafi ad un problema tempo - dipendente*, Politecnico di Torino.

[4] Mark Garcia, *Space Debris and Human Spacecraft*, NASA, 26 may 2021.

[5] D.J. Kessler, B.G. Cour-Palais, *Collision Frequency of Artificial Satellites: The Creation of a Debris Belt*, NASA Johnson Space Center, Houston, TX, 1 June 1978

[6] Donald J. Kessler, Nicholas L. Johnson, and J.-C. Liou, and Mark Matney, *The Kessler syndrome: implications to future space operations*, Advances in the Astronautical Sciences, Vol. 137, Univelt, San Diego, 2010.

[7] Pulliam, W., *Catcher's Mitt Final Report* Defense Advanced Research Projects Agency, 2011.

[8] Shen Hong-Xim, Casalino Lorenzo, *Simple ΔV Approximation for Optimization of Debris-to-Debris Transfers.*