TÉCNICO LISBOA

Monte Carlo Tree Search for Interplanetary Trajectory Design

Rita Fernandes Neves

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Rodrigo Martins de Matos Ventura
Co-Supervisor: Daniel Hennes

Examination Committee

Chairperson: João Manuel Lage de Miranda Lemos
Supervisor: Rodrigo Martins de Matos Ventura
Member of the Committee: Juan Luis Cano González

October 2015

Abstract

Interplanetary exploration represents a very small part of the space industry, since it involves many resources and its purposes are mainly scientific, as opposed to communications or surveillance. Currently, it is possible to make missions cheaper by including, on the spacecraft's course, maneuvers such as gravity assists (or flybys), that use the gravitational energy of bodies in space to reduce the acceleration needs.

In this way, trajectory design can be interpreted as a problem of finding the sequence of flyby bodies and times, from the Earth to the target planet, that meets our objectives and is limited by the given constraints. We can define two variants for this problem, depending on whether the times of flight are discretized or not. The first case is fully combinatorial, for both the flyby planets and times are discrete, while the second is hybrid. Nowadays, software tools for trajectory design have to deal with an extremely big search space of possibilities, which is why there is the need for a fast, computationally cheap algorithm to do so.

The Monte Carlo Tree Search is an algorithm that balances the exploration and exploitation of the state space in a way that very good results can be obtained in a short time and computational cost. We establish the most adequate constraints, experiment with four different Tree Policies regarding the learning of the method and study it both in the combinatorial case, applied to the search tree, and the hybrid case, combined with two different continuous optimization tools.

Keywords: Interplanetary Trajectories, Space Mission Analysis, Search Algorithms, Monte Carlo Tree Search, Optimization

Resumo

Exploração interplanetária representa uma pequena parte da indústria espacial, dado que envolve muitos recursos e os seus objectivos são meramente científicos, em vez de comunicações ou vigilância. Hoje em dia, missões podem tornar-se mais baratas ao incluir-se, na trajectória da nave, manobras como assistências gravitacionais, que aproveitam a energia gravítica dos corpos celestes para reduzir as necessidades de aceleração.

Desta forma, o design de uma trajectória pode ser interpretado como um problema de elaborar uma sequência de planetas usados para as assistências gravitacionais e tempos, da Terra ao destino, que cumpre os nossos objectivos e está limitada por certas restrições. Podemos definir duas variantes para este problema, consoante se aplica uma discretização aos tempos de voo ou não. O primeiro caso é inteiramente combinatório, uma vez que os planetas e tempos são discretos, enquanto que o segundo é híbrido. Hoje em dia, programas de software de design de trajectórias espaciais têm de lidar com um espaço imenso de possibilidades, criando a necessidade de implementação de um algoritmo rápido e computacionalmente leve.

O algoritmo de procura *Monte Carlo Tree Search* consegue balancear a exploração e o aproveitamento de boas possibilidades de uma árvore de procura, de forma a encontrar rapidamente resultados de qualidade. Neste trabalho, estabelecemos as restrições mais adequadas ao problema, experimentamos quatro *Tree Policies* diferentes e estudamos o caso combinatório, aplicado à árvore de procura, e o híbrido, utilizando o algoritmo combinado com duas ferramentas de optimização contínua distintas.

Palavras-chave: Trajectórias Interplanetárias, Análise de Missões Espaciais, Algoritmos de Procura, Monte Carlo Tree Search, Optimização

Contents

List of Tables

List of Figures

Nomenclature

Greek symbols

Δv	Change in velocity.
ρ	Regret.
τ	Orbital period.

Roman symbols

a	Acceleration.
a_H	Semi-major axis of an Hohmann Transfer.
g_0	Gravitational acceleration.
I_{SP}	Specific impulse.
l	Grid resolution parameter.
m	Mass.
P	Planet.
r	Position.
T	Thrust.

Subscripts

P	Planet.
S/C	Spacecraft.

Superscripts

$\ddot{}$	Second derivative.
$\dot{}$	First derivative.
$\vec{}$	Vector.

Chapter 1

Introduction

1.1 Motivation

Space exploration is a relatively new business field, considering the history of humanity. It started developing very quickly in the sixties, triggered by the events of the Cold War. At that time, the advances in the space sector were mostly achieved due to large investments made by the US and Russian governments. Nowadays, only a few more nations have managed to contribute to the space industry, and even fewer were capable of successfully carrying out interplanetary missions.

However, the space business has not become more accessible in any way, due to the costs involved. Currently, there are many scientific projects applying to few opportunities for fulfillment, as the funding continues to be scarce compared to the scope of the endeavors. Being so, to make a space mission as cheap as possible is the fundamental goal of every design.

When developing a space mission, one of the first things to be considered is the trajectory design. The path traversed from the Earth to the destination, especially when very far away, has to be carefully computed in order to fit the constraints: the launch window, the arrival time, the total fuel consumption and other demands that can be posed (for instance, the need to visit a certain asteroid along the way to get images of it).

Currently, it may be infeasible to consider a direct orbital transfer from the Earth - depending on the target planet, this could demand an enormous fuel consumption that would shoot the mission's budget to astronomical levels or pose inadequate time constrains in terms of launch and arrival times. Being so, the design is mostly made considering a series of gravity assist maneuvers to several astronomical objects (swing-bys or flybys - the first ones tend to be used for planets, flybys for other celestial bodies) during the entire trajectory. In this way, the spacecraft can harness the gravitational energy of these bodies to reduce the fuel needed for the journey and, therefore, minimize the total costs.

The problem remaining is that, depending on the goal, one can choose between many options for the flyby bodies, launch and arrival times. Thus, to find a sequence that obeys the posed constraints and minimizes the amount of required fuel becomes an optimization problem that can either be combinatorial, if the time intervals are discretized, or continuous, if they are not. The former can be treated as a search

problem; for the latter, optimization techniques should be implemented.

Nowadays, several software programs employ broad search and optimization methods capable of designing feasible trajectories, but this is a really demanding problem in terms of computation costs, considering the number of constraints and design parameters involved. Most of the methods employed nowadays, besides not being available to the public, rely heavily on heuristics and domain knowledge, which makes the evaluation of possible trajectories slow and complex. In this way, the development of a search method that evaluates these options efficiently and quickly without relying on domain specific information is of fundamental importance. For this purpose, we consider the application of the Monte Carlo Tree Search, which is a fast, stochastic algorithm that does not utilize any heuristic. It has been recently developed and tested mainly for two-player games, such as Go ([1]), so there are still many areas in which this algorithm can be applied.

On the other hand, if we treat this as an optimization problem, it has to be studied in terms of convexity and, depending on this characteristic, different optimization techniques can be implemented. A full optimizer of interplanetary trajectories has already been developed in [2], but it employs proprietary software by TUDelft which is not available to the general public; in contrast, this project is built only on open source code.

We can then divide this subject in three parts: the Model used for this problem, detailing the Orbital Mechanics involved; the Combinatorial Optimization part, regarding the search for the best solution; and the Hybrid Optimization, dealing with the integration of Monte Carlo Tree Search with optimization methods.

1.2 Problem Statement

The purpose of this project is to test the Monte Carlo Tree Search algorithm as an efficient, fast method to determine an interplanetary trajectory which obeys several placed constraints on the mission. The design purposes can be related to minimizing the total Δv (proportional to the fuel consumption) or maximizing the number of celestial bodies observed for science.

This can be done with combinatorial optimization, in which the time windows are discretized, or with a hybrid kind of optimization, fusing the discrete nature of the planet search with the continuous launch dates. We propose also to compare the performance of the Monte Carlo Tree Search algorithm on both these frameworks.

In the combinatorial optimization scope, this problem can easily be converted into a tree search by considering some important parameters. These are the initial time of departure from the first planet (which is, for simplicity's sake, the Earth), the bodies visited and the times of flight to them associated. A planetary trajectory can be described by a string of the following parameters: $\{P_0, T_0, P_1, T_1, ...\}$ (as explained in [3]). In here, T_0 is the departure time in Modified Julian Date 2000 (MJD2000), while the other times are intervals of flight in between two consecutive planets.

Therefore, there are three kinds of nodes to be considered: Planets/Bodies, which can be taken out of a discrete pool of choices; the launch time T_0, limited by an imposed launch window; and the times of

flight (tof), which depend on the departure and target planet for that flyby - for instance, farther away planets will possess a bigger interval for the sampling of times of flight than ones that are closer.

1.3 Contributions

The framework for this thesis was already started in [3] by Dario Izzo and Daniel Hennes; the remaining work developed was done in collaboration with the European Space Agency.

In this way, new methods were developed in order to perform parameter tuning for the implementation of the four tree policies. Distinct budgets were used for comparison purposes, and all the code was written with the objective of making it more efficient and faster - for the analysis of the different code segments and how they could be optimized, Python tools were utilized. The hybrid problem was considered, both on the HOOT side and the coupling with optimization techniques. Finally, the Asteroid mission problem was studied, from the consideration regarding its time discretizations to the application of the optimization methods.

1.4 Literature Survey

Interplanetary space missions represent a small part of the current budget of space agencies worldwide, since the public demands tend more towards communication and observation satellites, which orbit around the Earth. Manned interplanetary spaceflight has not yet been achieved, but robotic and probe exploration are delivering breakthrough scientific discoveries. As examples, we have the first spacecraft employed in an interplanetary mission, the Venera 1 in 1961 ([4]), the first probe to leave the Solar System, the Voyager 1 ([5]) and New Horizons, arriving to Pluto in 2015 ([6]).

One of the most important challenges to tackle regarding this problem has always been the trajectory planning, since the accelerations involved in space travel demand a lot of fuel and the parameters involved in a mission change with the planetary positions. In order to design a trajectory, several software programs were created that could model the environment and physics of space. On top of them, search algorithms and optimization methods were implemented, so that the final spacecraft's course could be computed regarding certain mission objectives.

Worldwide, space agencies use different orbital mechanics simulation software and, sometimes, more than one simultaneously. For instance, explored by NASA (the National Aeronautics and Space Administration, from the United States), we have: the Johnson Space Center Engineering Orbital Dynamics (JEOD, [7]), a software that generates vehicle trajectories and also includes environment and dynamics models for the dynamic state of a spacecraft in a planetary environment; the Copernicus ([8], [9]), a spacecraft trajectory design and optimization tool; the General Mission Analysis Tool (GMAT, [10]), an open-source space mission analysis software with broad resources that allow for modeling, optimization and estimation of spacecraft trajectories; the Mission Analysis Low Thrust Optimization (MALTO), an optimization tool for low-thrust routes; the Satellite Tour Design Program (STOUR, [11]), for the complete analysis of several planetary sequence possibilities in trajectory design. ESA (the European Space

Agency) utilizes the Space Trajectory Analysis (STA, [12]), together with the Systems Toolkit (STK), which possesses many potentialities, such as orbital simulation (as can be explored on [13]).

These are not, however, the only available tools for interplanetary trajectory design; more continue being created and developed every year, and many are even available to the general public. However, most of them take a long time to evaluate the trajectory options and also rely heavily on domain information (notably, the STOUR software, employed in several missions, as in [14]). Following this, the Monte Carlo Tree Search is a recently developed algorithm that may be employed on mission design for faster results.

Monte Carlo Tree Search combines the randomness of Monte Carlo simulations with tree search and is typically applied to combinatorial problems. The method was first formalized in 2006 ([15]), many years after the first developments of Monte Carlo simulations and tree search in the early 20^{th} century. On the last few years, its development has grown due to its performance in a multitude of games, especially Go.

Go is a traditional dual-player board game, played on a 9×9 or 19×19 board - naturally, the bigger it is, the more complex the game ends up being. The goal of the game is to conquer the biggest part of the board, by putting stones on it and capturing the opponent's. Nowadays, it stands as the benchmark for the development and enhancement of algorithms in Artificial Intelligence, having substituted chess, whose players are already fairly advanced (IBM's Deep Blue beating Gary Kasparov [16], Stockfish chess engine [17]). In Go, however, branching factors are orders of magnitude larger and useful heuristics for minimax or $\alpha - \beta$ search are much more difficult to formulate.

Nowadays, Monte Carlo Tree Search, which is heuristic-free, has been applied very successfully to Go, developing it from strong beginner level (minimax algorithm) to beating a professional player on a 9×9 board in 2008 ([18]) and on a 19×19 board with a nine stone handicap ([19]). There are several other applications for this algorithm, such as connection games like Hex ([20]), real-time games like Tron ([21]) and single-player games like Morpion Solitaire ([22] and [23]). Our specific problem can also be interpreted as a single-player game, although the domain is different than the one described.

Chapter 2

Model

2.1 Introduction

Interplanetary trajectory design can only be done using a reliable framework in which we are able to simulate the true physics of space travel. This is done using the software PyKEP ([24]), an open source project coded in `C++` and exposed to Python. This software allows us to compute the parameters involved in a series of celestial maneuvers and, in this way, simulate mission trajectories, all in conformity to the laws of space dynamics.

This chapter intends to describe the orbital mechanics and maneuvers involved in the design of a mission trajectory. Furthermore, it will detail the test cases for which we intend to apply our algorithms to: each of them is a separate mission with different purposes and constraints.

2.2 Orbital Mechanics

The motion of celestial bodies in the Universe has been a subject of study for centuries. From the observation of the position of the stars in order to determine the time of the day, to space travel and the formation of satellite constellations, Humanity has come a long way. We have built thousands of spacecrafts designed to leave the Earth's atmosphere: however, only a small number of them was intended to exit this planet's orbit.

In order to design a space mission, one must first understand the physical principles behind planetary motion, especially the Kepler's laws, mathematically derived from Newton's laws (most of the time, the relativistic effects found by Einstein are not considered). Generalizing Kepler's first law, it can be said that all orbits have the shape of a conic section: the simplest ones are ellipses, with the planet that exerts the gravitational pull at one of the foci, as we can see on Figure 2.1.

We can consider this as a model for the motion of the Earth around the Sun; the latter would be in F and the former moving along the position vector r.

One of the main questions when designing a space mission is the influence each celestial body has on a spacecraft. It is only possible to numerically solve gravitational problems regarding the influence of one

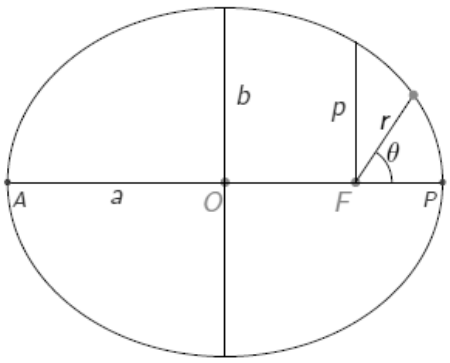


Figure 2.1: Elliptical Orbit. Semi-major axis a, Semi-minor axis b, Focus F, Periapsis P and Apoapsis A.

body on another - this is called a two-body problem and is a classical question in Physics. In this way, space is divided in spheres of influence, one for each planet or star: when inside one, the ship is affected only by its gravitational pull.

2.2.1 Orbital Transfers

The planning of an interplanetary trajectory from the Earth to a target planet is a matter of designing an orbital maneuver. This should be done by employing the smallest amount of fuel possible, which is preponderant in the mission cost. For this, we use the parameter Δv: this quantity is proportional to the thrust per unit mass, which can be used to determine the quantity of propellant required for the given maneuver using the Tsiolkovsky rocket equation below. Physically, it represents the difference in velocity of the spacecraft computed between the initial and final points of the maneuver.

$$m_2 = m_1 \exp \frac{-\Delta v}{g_0 I_{SP}} \tag{2.1}$$

in which m_1 and m_2 are respectively the initial and final mass, g_0 is the standard acceleration due to gravity and I_{SP} is the specific impulse, characteristic of every engine.

Therefore, the design of an orbital transfer has to do with the determination of the orbit that goes through two specific points. For instance, if we need a trajectory between planets, these points in space will be the *ephemerides* of the two bodies (natural positions occurring at a given date). The task of finding the correct orbit knowing only these position and time vectors is called the *Lambert's Problem*. Physically, this consists on the boundary solution for the differential gravitation equation of the two-body problem below:

$$\ddot{\vec{r}} = -\mu \cdot \frac{\hat{r}}{r^2} \tag{2.2}$$

With this equation, the Lambert's problem becomes to find solutions for the position vector $\vec{r}$ in

which $\vec{r}(t_1) = \vec{r_1}$ and $\vec{r}(t_2) = \vec{r_2}$.

We call a *Lambert leg* to a solution to the Lambert's problem. Thus, to compute an orbital maneuver between planets, we need the Lambert leg that connects the ephemerides of these two bodies at given times. Using this, it is possible to find the velocities for the spacecraft at the beginning and end of the trajectory, as we can see on Figure 2.2, employed to determine the Δv needed to perform the maneuver.

In this way, an interplanetary trajectory is composed of one or more Lambert legs, depending on whether it is a *direct transfer* between two planets or a series of *gravity assists*, where several planetary swing-bys may happen.

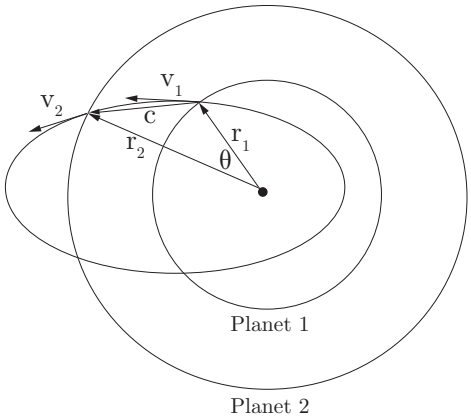


Figure 2.2: Lambert Leg between Planets 1 and 2

In PyKEP, each Lambert leg is computed as detailed in [25]. This software utilizes the SPICE database from the Jet Propulsion Laboratory ([26]) to get the position vectors of each planet at a given time, and uses this information plus the time of flight to compute the incoming and outgoing velocities from the two celestial bodies, which will be used for the calculation of the maneuver's Δv.

Direct Transfers

Regarding space missions, there are some predefined direct orbital maneuvers which have been studied over the years in much detail. One of them is the **Hohmann Transfer**, developed in 1925 ([27]), which is the direct interplanetary trajectory that requires the least amount of propellant. This orbital maneuver consists on an elliptical orbit whose periapsis is contained in the departing planet's orbit (planet A), whose apoapsis is on the goal planet's orbit (planet B) and whose focus is the Sun, as we can see on Figure 2.3.

Thus, there are two Δv required for the entirety of the transfer orbit: a positive one, resulting from the difference between the Hohmann and the departing planet's orbits Δv, and a negative one, consequence of the difference between the goal planet's and Hohmann orbits Δv. The total amount of Δv required is the sum of these two, as described by the following equations:

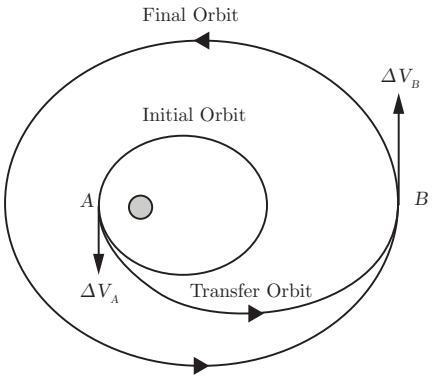


Figure 2.3: Homann Transfer from point A to point B, from [28]

$$\Delta v_A = \sqrt{\mu\left(\frac{2}{r_A} - \frac{1}{a_H}\right)} - \sqrt{\frac{\mu}{r_A}} \tag{2.3}$$

$$\Delta v_B = \sqrt{\frac{\mu}{r_B}} - \sqrt{\mu\left(\frac{2}{r_B} - \frac{1}{a_H}\right)} \tag{2.4}$$

$$\Delta v = \Delta v_A + \Delta v_B \tag{2.5}$$

However, for this maneuver to be successful, the launch window is extremely narrow - the ephemerides of the planets have to be perfectly aligned so that the spacecraft arrives precisely at the point where planet B is, as it can be seen on Figure 2.3.

Gravity Assists

A direct orbital transfer such as the Hohmann maneuver is not the best option for a long mission, since the necessary Δv may be excessive and the time constraints too restrictive. Thus, most interplanetary trajectories rely on a series of gravity assists ([29]). Each one consists on a flyby of a body to use its angular momentum, which decreases the fuel consumption. Therefore, the spacecraft's velocity can be changed in whichever way is needed just by varying the angle and direction of approach to the planet. The principles behind this maneuver are the conservation of the kinetic energy and of the linear momentum of both bodies, which can be seen on the following equations:

$$m_{S/C}\Delta v_{S/C} + m_P \Delta v_P = 0 \tag{2.6}$$

$$\frac{1}{2}m_{S/C}\Delta v_{S/C}^2 + \frac{1}{2}m_P \Delta v_P^2 = 0 \tag{2.7}$$

From equation 2.6, the linear momentum gained by the spaceship is equal in magnitude to that lost by the planet, but these effects are ignored in the calculation since the planet is much more massive than the spacecraft; thus, Δv_P is negligible. The final velocity V_{out} reached by the spacecraft is determined by vectorially adding the hyperbolic excess velocity $V_{\infty_{out}}$ and the target planet's velocity V_{planet}, according

to the following equation:

$$\vec{V_{out}} = \vec{V_{\infty_{out}}} + \vec{V_P} \tag{2.8}$$

If the angle between the spacecraft and the planet is zero, the equation becomes a scalar sum in which the final velocity is equal to the initial one plus two times the planet's speed (derived from equations 2.6 and 2.7). This can be seen on Figure 2.4.

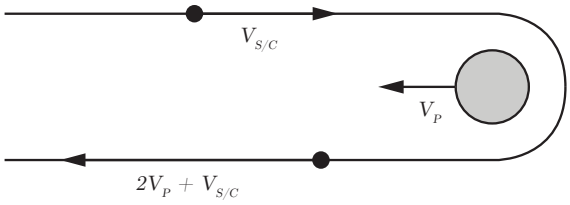


Figure 2.4: Gravity assist approach with angle zero

A well-established way to get more energy from a gravity assist is to fire a rocket engine at the periapsis, where a spacecraft is at its maximum velocity. This is called a *powered gravity assist*, which is based on the **Oberth effect**. This law determines that the use of a rocket engine when traveling at high speed generates more useful energy than at low speed.

2.2.2 Chemical Propulsion versus Low-Thrust

The discussion of interplanetary trajectories so far is based on the use of chemical rockets, in which a launch vehicle provides nearly the whole of the spacecraft's propulsive energy. All of the available fuel comes from the Earth and, therefore, the amount to be spent on maneuvers is limited. This way, the spacecraft may fire short bursts from its chemical rocket thrusters for small adjustments in trajectory very sporadically but, otherwise, it is in free-fall or using the energy harvested from gravity assists.

However, in recent years, electric propulsion has been developed. This energy can be harnessed from the Sun (Solar Electric Propulsion, or SEP) or from a nuclear source (Nuclear Electric Propulsion, or NEP), when the first is not available - for instance, in the case of deep space missions. Instead of functioning with small bursts of high power, the electric propulsion engines work with small amounts of thrust, which therefore have to be more frequent in time (almost continuous), making the engines operate for longer periods. This offers ten times more efficiency than chemical propulsion (higher specific impulse) and, in contrast, there is no limit in energy, only in power - meaning we can harvest as much energy as necessary, but the thrust obtained is much smaller, since the rate at which this energy can be supplied depends on the mass of the propellant.

Thus, electric propulsion is being used increasingly often for station-keeping, orbit raising and as a primary means of propulsion. It is still fairly undeveloped and, at this state, it is not possible to use it to launch from the Earth - currently, spacecrafts that have the capability to keep both systems are the most common.

In our model, it is possible to treat a Lambert leg as an impulsive transfer, with chemical propulsion, or as a low-thrust one. The low-thrust model uses NEP and was developed by Landau ([30]); it delivers a and Δv values adequate for this kind of maneuver, given the velocities yielded by the Lambert leg computation. It is important to observe that, while utilizing low-thrust impulses, each spacecraft is limited by a weight threshold, above which the maneuver cannot be done. The model can also be used to compute this value, which we call m^* - a Lambert leg is infeasible if m^* is smaller than the current spacecraft mass. This calculation is done by the following equation:

$$m^* = \frac{2T_{MAX}}{a\left(1 + \exp\frac{-\Delta v}{g_0 I_{SP}}\right)} \tag{2.9}$$

in which m^* is the threshold mass value, T_{MAX} is the maximum thrust, a is the spacecraft's acceleration, I_{SP} is the specific impulse and g_0 is the gravitational acceleration.

2.3 Test Cases

It is easy to formulate several problems that request the design of an interplanetary trajectory: all we need is the departure and destination planets. However, these propositions should be related to real missions, so that the results can be compared to the solutions found by the actual designers. In this way, we have selected the Rosetta-Philae and Cassini–Huygens missions, two chemical thrust ones, as test cases. Another situation, considered afterwards, is the Asteroid problem, in which the goal is to visit as many bodies as possible in a limited period of time. In this case, the trajectory is considered to be low-thrust and the Δv is not the main mission parameter, so we do not deal with it directly.

2.3.1 The Rosetta Problem

One of the cases for the trajectory design is the Rosetta mission, developed by the European Space Agency. This problem consists on finding the optimal trajectory between the Earth and the comet 67P/Churyumov-Gerasimenko, the final destination of the probe. We consider the best trajectory to be the one that spends the least Δv, which corresponds to the sum of the Δv of each Lambert leg in the sequence.

The real trajectory of the Rosetta mission spanned a period from March, 2004 to May, 2014 (the rendezvous with 67P happening on November of the same year). The planetary sequence traversed was composed of a series of swing-bys to Earth with an initial Δv-EGA maneuver ([31]), Mars and Earth again, a flyby to the asteroid 2867 Steins, a final swing-by with the Earth and a flyby with the asteroid 21 Lutetia. This trajectory was not chosen only for its minimization of the Δv budget, but also for scientific purposes (imaging of the asteroids), and therefore may not be the best one as evaluated by the criteria used on this project.

The considered mission depends purely on chemical thrust, and the flyby planets are Venus, Earth, Mars and Jupiter, which are the closest ones to the asteroid. The constraints in time of flight windows

depend on the distance between planets.

2.3.2 The Cassini Problem

The purpose of this mission was to send an unmanned spacecraft to orbit the planet Saturn. The probe launched on October, 1997 and got into the orbit of Saturn on July, 2004. Cassini contained a lander for the Titan moon, called Huygens, which landed in 2005, being the first one deployed successfully on the Outer Solar System.

The real trajectory of the Cassini mission consisted on two swing-bys of Venus, one of the Earth, Jupiter and, finally, the orbital insertion. In total, the travel time took around 2480 days; the mission itself is expected to end on 2017.

Our problem is then to find the mission trajectory that spends the least Δv. This mission is also considered to be chemical thrust, with the same swing-by planets as the Rosetta one, but Saturn as target.

2.3.3 The Asteroid Problem

The purpose of this problem is to develop a path that, with a limited time duration, fixed starting spacecraft mass and launch date from a certain asteroid, allows for the spacecraft to visit as many asteroids as possible. These types of trajectory design have been increasingly more studied in the later years, due to the recent development of NEP systems ([32], [33]).

This problem is adapted from the formulation of the 7^{th} edition of the Global Trajectory Optimization Competition (GTOC, [34]). The considered mission is a low-thrust heliocentric one. The asteroid search space has a fixed size of 67 bodies (including the departing one) and we consider that the probe launches from a fixed asteroid, *Mimosa*, but it cannot visit the same body twice. There are two conditions to a path's termination: the journey must not take longer than six years to complete and the vehicle mass for a given Lambert leg must be smaller than m^*, as in Equation 2.9.

We consider that the spacecraft departs from the first asteroid with an initial total mass of 2000 kg on 8935 MJD2000 and must stay at least 30 days on each asteroid. If no termination condition is reached, the new visited asteroid is added to the path and the new spacecraft mass is computed using the Tsiolkovsky formula in Equation 2.1.

In this way, the search space is very broad, even more so than on the previously considered problems. The evaluation function changes from the other test cases: the purpose of Rosetta and Cassini is to find the trajectory that spends the least Δv, and this is to discover the path of maximum length - the biggest branch of the tree.

Chapter 3

Combinatorial Optimization

3.1 Introduction

As considered on Section 1.2, we have established that the interplanetary trajectory design problem can be expressed as a tree search. This tree is detailed on Figure 3.1: we can see there are three different types of nodes (representations of the problem's state) required: the launch date, the swing-by planets and the times of flight in between. The last two of them repeat themselves in sequence until the target planet and arrival date are found. Having this problem defined in this way allows for the implementation of search algorithms to find the best solution contained in the structure.

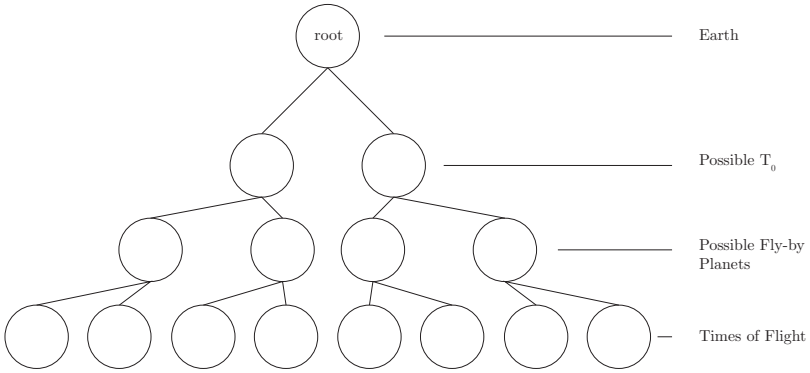


Figure 3.1: Interplanetary Search problem in the form of a Search Tree

The following sections explain in detail the basics of search methods and algorithms, their adaptability to this specific problem and how to convert the time intervals into discrete grid points. Furthermore, we will detail the implementation of Monte Carlo Tree Search on this combinatorial problem and the results obtained.

3.1.1 Time Discretization

Considering the interval of each time of flight to be continuous, a discretization of the time states is a useful approach in order to convert the entire problem to a combinatorial one. After establishing an

appropriate grid resolution, the entire problem becomes combinatorial and discrete - thus, several tree search algorithms can be used in order to find the best solution.

Rosetta and Cassini Problems Since we are considering the times of flight between very differently located planets, it is easy to discern that a uniform grid is infeasible in this case: we cannot sample the launch window between Earth and Jupiter (which is very big) in the same way as we do it for the planets Earth and Venus.

The grid must then sample all of the possible intervals in the same way, allowing for the same branching factor. This is done by a discretization method that takes into account the ephemerides and distances between the planets, as described in [3]. This discretization is represented by Equation 3.1:

$$G = \{t_0 + \frac{jl\tau_i}{360^\circ}, j \in \mathbb{N}\} \tag{3.1}$$

Here, $l = 11.25^\circ$ is the smallest grid resolution considered, following the results of [3] and j is the point resulting from the piecewise decomposition.

Asteroid Problem Since there are many asteroids in the search space whose distances in between are not known but that are reasonably in the same orbit, the time discretization is not made specifically for each pair of them as in the previous test cases: it is the same for all. Even so, different variations of the grid were implemented, as follows:

- **Linear Grid**: from 60 to 330 days, evenly spaced in intervals of 30 days.
- **Logarithmic Grid**: also from 60 to 330 days, with a greater concentration of days in the middle point of these extremes.
- **Forced Grid**: Similar to the linear one, but the algorithm is always forced to choose the smallest possible time of flight, so that maximum mission time constraint is reached as late as possible. In this way, it starts by choosing the value 60 for a given asteroid; if it is not feasible, goes on to 90 days, and so on.

The first two discretizations are illustrated on Figure 3.2; the Forced one is not, since it is the same as the Linear in terms of the time intervals.

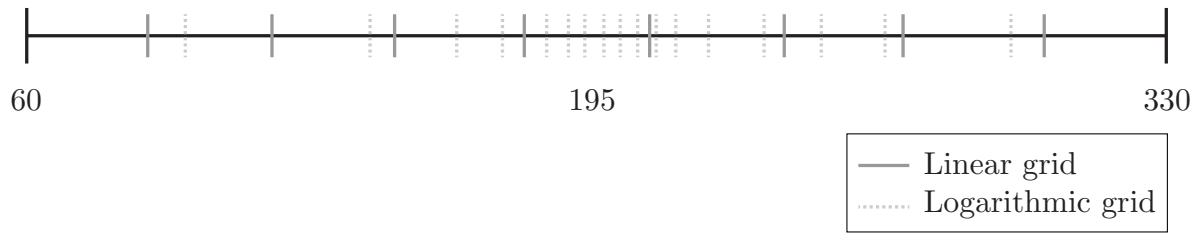


Figure 3.2: Linear and Logarithmic Time Grids for the Asteroid Problem

3.1.2 Search Methods

The search methods to be applied on this problem are employed on a domain in the form of a tree, which means they will be developed in the form of a tree search.

A tree is a data structure designed to efficiently organize information in a hierarchical fashion. It consists of a set of linked *nodes*: each one is a data structure containing a value or characteristic, with references to the ones below. On the top of the tree, there is the node that contains the initial information, called the *root*; all the nodes connected to it are its *children*. In the same way, we can also consider the existence of parent nodes - while one node can have many children, one cannot possess more than one parent. The height of the tree is considered its *depth* and its width is called *breadth*; nodes can be appended to the tree both in depth and in breadth. The depth of the tree is equal to the number of connections between the node and the root, plus one (if we consider the root to be at the first level).

A *path* is the sequence of nodes traversed from the root to the one we are considering. Finally, a *leaf* node is one that has no children. The tree is traversed from the root node downwards, until the leaf.

There are several problems that benefit from the representation as a tree. The solution ends up being a sequence of states and the moves in between them; the root is the initial state, a certain node is the goal one and the moves are the branches that connect them. In order to find a specific node or path on the tree, one has to employ a search algorithm. Tree search algorithms have been studied since the 19^{th} century ([35]), and are generally classified by the order in which the nodes are visited. These methods can have an *heuristic function* associated, which gives additional information about the best next nodes to explore.

3.1.3 Problem-Specific Heuristics

The use of a heuristic-based algorithm would make the search for the best solution much more effective than it currently is. For instance, one of the most studied and implemented algorithms for tree search is the $A*$, introduced in [36]. Its *evaluation function*, which qualifies the next nodes to explore, is the sum of the path cost $g(n)$ traversed so far and the heuristic of the problem $h(n)$, as on the following equation:

$$f(n) = g(n) + h(n) \tag{3.2}$$

A heuristic function should be both *consistent* and *admissible*, the latter meaning that it should never overestimate the true cost to the goal. However, this is not easy to determine for this problem - the Δv budget to the goal planet cannot be underestimated in a different way for each possible body in the search space. If a lower bound was found, it would be the same for every planet, due to the fact that the possibility to perform the swing-bys makes it that no node is better than the other. For instance, we can consider as a possible heuristic function the Δv budget computed for a direct transfer (for instance, a Hohmann maneuver). However, although this is different for each planet, it would not consist on an underestimation - otherwise, the best solution for the interplanetary trajectory problem would be a

direct transfer from the departing to the target planet. If an underestimation was easy to determine, the smallest Δv solution would also be so.

Thus, it is very hard to find a suitable heuristic for this problem. The alternative is to consider a heuristic-free algorithm that remains fast and reliable. Both the exhaustive methods applied in the following Section and the Monte Carlo Tree Search are, therefore, heuristic-free.

3.1.4 Software

This specific problem requires the realistic knowledge of the ephemeris of each celestial body at a time epoch, as well as a means to calculate several characteristics of an orbital transfer maneuver between two planets, such as the Δv. The model we use for this representation is the scientific library PyKEP. The algorithms utilized are all coded in the Python programming language.

3.2 Preliminary Implementations

To establish what the best solution contained in the tree structure is, the one that will serve as the benchmark, an exhaustive search method has to be implemented. The tree can only be successfully searched in this way if it is finite, since we are limited by time and computational resources. To do so, constraints were applied to the problem in terms of mission duration and maximum Δv reached for each solution. The latter one can be any value greater than the best solution (since, at this point, we only care about the smallest Δv) and the maximum mission times were considered to be 4000 and 3000 days for the Rosetta and Cassini cases, respectively (a rounding of the real mission times mentioned in Section 2.3).

Being so, established and thoroughly studied search methods were applied to the problem and the results can be found in the following subsections.

3.2.1 Exhaustive Search

Rosetta and Cassini Problems

Since the tree is finite due to the pruning of nodes whose path reaches a certain Δv and maximum mission time, it is possible to implement the method of the depth-first search, which in this case is optimal and complete, running in a finite amount of time and always finding the best solution. The space complexity of this method is of $O(b.d)$, which is smaller than other exhaustive search methods such as the breadth-first search, obtaining the same results with smaller computational cost.

The best solutions taken from a run of this search method are described in Tables 3.1 and 3.2 for both the Rosetta and Cassini problems. We have considered a previously established starting date, as indicated on both tables, which reduces our search space - this had to be done, since otherwise the algorithm takes longer to run than what would be acceptable. All the values were rounded to two decimal places.

We now have the best results for this discretized search tree, which are in fact very similar to each other - both paths start with the sequence EVVEJ (using only the initial of each planet for simplification

T_0 (MJD2000)	1574.14
Planetary Sequence	['Earth', 'Venus', 'Venus', 'Earth', 'Jupiter', 'Earth', '67P']
TOF Sequence (days)	[159.71, 435.36, 55.54, 725.31, 655.82, 1531.98]
Total Δv (m/s)	4021.93
Total Computed Legs	3824185323

Table 3.1: Best flight sequence for the Rosetta Problem

T_0 (MJD2000)	-785.86
Planetary Sequence	['Earth', 'Venus', 'Venus', 'Earth', 'Jupiter', 'Saturn']
TOF Sequence (days)	[194.82, 400.25, 55.54, 589.85, 1670.81]
Total Δv (m/s)	8525.14
Total Computed Legs	4025526970

Table 3.2: Best flight sequence for the Cassini Problem

purposes), from which we can infer that, generally, this is a small Δv sequence. Compared to the real mission sequences, described in Section 2.3, we can see that the Cassini one actually corresponds to the real conducted one, which gives strength to our model and the discretization. The Rosetta one is, expectedly, different, since the actual path included one Δv-EGA maneuver not available in our model.

It is important to denote that the Cassini problem has a bigger search tree than the Rosetta one, induced from the number of Lambert legs that were computed in total. Considering that both these searches took very long to compute (in the order of weeks), a method that could find the optimal solution without this great computational cost would be very useful. Besides, mission design is an extremely volatile process and it happens often that launch dates have to be postponed, meaning the trajectory has to be computed again with short notice. Thus, a tentative implementation of Beam Search was made, in order to compare the results of both methods and check if the best solutions could be obtained faster.

Asteroid Problem

For the Asteroid case, it was impossible to explore the entire tree in an acceptable time span, since the size of the problem is even bigger than the previous one. Therefore, the results for the exhaustive search are not here presented.

In this way, the baseline result used for comparison purposes is the biggest sequence found on the 7^{th} edition of GTOC - 13 (solution of 14 asteroids is referenced, but never divulged, in [34]). The methods employed in this results varied from Ant Colony and Particle Swarm Optimization to Genetic Algorithms, coupled with the utilization of MALTO software. However, it is fundamental to denote that this value was found with different starting asteroid, launching time and bigger pool of celestial bodies to choose for the path.

3.2.2 Beam Search

Beam search is a method similar to breadth-first search which, at each depth level of the tree, takes a fixed amount of nodes and discards the rest, reducing the branching factor of the search. The expanded

nodes are the ones considered to be the best so far and end up composing the 'beam' that gives the name to this method. This evaluation is done regarding the value of the accumulated Δv of the node - the smaller this is, the greater the node's quality.

Being so, this implementation is considered a type of best-first search, since only the path cost up to the current depth level is considered and no prediction of the upcoming cost is made - no heuristic function is here applied. Therefore, this algorithm presents itself as an alternative to the previously implemented method, with the potential to deliver the results in a shorter time period.

Rosetta and Cassini Problems

For these cases, two variants of the algorithm were considered: one that keeps as many best nodes as the pre-determined beam size, and another that does the same, but the best nodes are distributed into different swing-by planets. Practically, while the first algorithm keeps the overall N best, the second does so for N/S best nodes of each planet, S being the number of different planets that exist for the swing-by. The latter one is expected to yield a better performance, avoiding repeated states and getting trapped in costless sequences such as ['Earth', 'Earth', 'Earth']. The results from these two can be found on Table 3.3.

		Beam Size					
Algorithm	Parameter	10	100	1000	10000	100000	1000000
1	Δv (m/s)	6686.46	6686.46	6686.46	4469.26	4469.26	4021.32
	Computed Legs	7817	26366	209180	2522404	21729392	55916093
	Complete?	No	No	No	No	No	Yes
2	Δv (m/s)	6686.46	6686.46	6686.46	4469.26	4469.26	4021.32
	Computed Legs	7817	26800	209180	2522404	21729392	55915948
	Complete?	No	No	No	No	No	Yes

Table 3.3: Beam Search results for the Rosetta problem with different Beam Sizes

		Beam Size					
Algorithm	Parameter	10	100	1000	10000	100000	1000000
1	Δv (m/s)	12448.38	12448.38	8457.64	8457.64	8457.64	8457.64
	Computed Legs	4227	18728	55826	413781	2814835	7059998
	Complete?	No	No	No	No	No	No
2	Δv (m/s)	12448.38	8457.64	8457.64	8457.64	8457.64	8457.64
	Computed Legs	12811	28390	245506	1438556	7059945	7059945
	Complete?	No	No	No	No	No	No

Table 3.4: Beam Search results for the Cassini problem with different Beam Sizes

As we can see, the second algorithm performs slightly better than the first one, but not by much; on the Rosetta case, it requires less computed legs to reach the optimal outcome but always finding the same Δv value with these beam sizes; although the amount of computed Lambert legs is ten times smaller than the one of the depth-first search, the best result is found only when the beam is extended to one million nodes. On the Cassini one, it arrives to different Δv with a small beam, but it never reaches the optimal solution.

It is possible to infer that the poor results are due to the fact that the algorithm is choosing the nodes it preserves according only to a cost of the path traversed so far, which does not contain any information about the upcoming cost to the goal state. For this, an heuristic function would be needed as an estimation of the true cost between the current node and the goal node. This is, however, not possible to implement, as previously determined.

Asteroid Problem

For the asteroid case, we have considered three different beams: one that chooses the nodes with the smallest mass, getting the spacecraft to be lighter and more maneuverable; other that selects the nodes with the smallest mission time spent, so that the time constraint is met later on; and finally, a beam that considers both of these goals to have the same importance. The sequence lengths obtained are shown in Table 3.5.

	Beam Size				
Variable	10	100	1000	10000	100000
Mass	3	4	10	10	10
Sum	4	4	10	10	10
Double	4	4	10	10	10

Table 3.5: Beam Search results with different Beam Sizes for the Asteroid Problem

As we can see, the distinct objective function produces almost negligible differences in the results in terms of sequence length. It seems that the one to perform in a slightly worse manner is the beam that saves states where the spacecraft's mass is the lowest, but this difference is only evident in very small beam sizes. As this parameter gets bigger, the algorithms seem to perform in the same way, needing only a beam of size 1000 to get to the best result possible.

This method is not expected to give us the optimal results - the values yielded are very far from the baseline. This is due to the fact that, for the case where nodes with the smallest times are saved, many useful states are eliminated at the beginning - sequences tend to start with a big time of flight and then proceed to shorter time periods, when the spacecraft is lighter. On the contrary, when nodes with the smallest mass are saved, these states may correspond to ones with a greater mission time.

In this way, beam search is not a fast and reliable alternative to the exhaustive search method presented.

3.3 Monte Carlo Tree Search

The Monte Carlo Tree Search is, as the name indicates, a tree search algorithm, but also a Monte Carlo method, a statistical scheme that relies on random sampling from a determined problem state, performing a computation from it and generating results. In this way, there is an unknown probability of finding the optimal solution, but that is not guaranteed: depending on the problem, if enough samples are taken, eventually the desired outcome will fall into the pool of visited states. It is an aheuristic algorithm, not

depending on domain specific knowledge, although it is possible to include information on it. It is also asymmetric, not exploring the tree uniformly, being biased towards more promising areas of exploration.

The general framework of the Monte Carlo Tree Search algorithm is presented on Algorithm 1. The method consists of four sequential steps that are repeated while a given budget lasts: *Selection*, *Expansion*, *Simulation* and *Update*. This budget can be defined in many ways: for instance, in terms of time or number of simulations performed.

The algorithm starts with only the root node: the first step, the *Selection*, consists of choosing which node should be played next, until a leaf is reached. This is done using a *Tree Policy*, which will be explained in detail in Section 3.3.1. Once this happens, we go through the *Expansion* phase to create a new leaf node, resulting from a random move taken on the previous node. The moves are the possible children of the current node: for instance, if we are at a planet node, the moves from there on are the possible times of flight to that planet.

From this point forward, we enter the *Simulation* step: moves are performed, using a Simulation Policy (the default chosen is random) until we achieve a terminal state, meaning we have reached the goal (target planet) or violated our constraints. At the end of this, a reward is computed to assess the quality of the reached state. Finally, we *Update* the statistics of each node traversed with the computed reward, while backtracking until we are again at the root.

```
While budget is not over
    Selection
        node ← node.Select(TreePolicy)
    end
    Expansion
        node ← node.Expand(RandomMove)
    end
    Simulation
        state ← state.Move(SimulationPolicy)
        If Terminal then
            break;
    end
    Backtrack
        node.Statistics ← Reward
    end
end
```

Algorithm 1: Monte Carlo Tree Search

3.3.1 Tree and Simulation Policies

There are two main aspects of the Monte Carlo Tree Search that can be changed for algorithm variability. One is the *Tree* or *Selection Policy* and the other one is the *Simulation* or *Rollout Policy* (a *rollout* is each move in the simulation, made from the tree to a leaf node).

The *Tree Policy* defines the choice and expansion of the nodes already contained on the tree while building it. While traversing it, they can be chosen at random or *confidence bounds* can be used to estimate the probability of the outcome being the optimal solution.

These confidence bounds are used to balance the *exploration* versus the *exploitation* of the search tree.

This is defined as the *Multi-Armed Bandit* problem and formulated in the following manner: we consider a gambler at a row of slot machines, where he has to decide which one of them to play, how many times to do so for each machine and in which order. When he pulls a lever, each machine provides a random reward. The objective of the gambler is to maximize the sum of rewards earned through a sequence of machine choices. In this way, the gambler can choose the machine he knows, from experience, to give the highest expected payoff (exploitation), or to experiment with other ones to get more information about them (exploration) and perhaps get an even higher result. The balance between exploration and exploitation is done in order to maximize the cumulative reward over the time period considered.

Therefore, the Multi-Armed Bandit problem relies on the elaboration of a policy to decide what is the next move to play in order to optimize the expected total reward and minimize the expected regret ρ - the difference between the sum of rewards associated with an optimal strategy $\mu^* n$ and the sum of the actual collected rewards after n plays. The regret can be expressed as seen on Equation 3.3:

$$\rho = \mu^* n - \mu_j \sum_{j=1}^{K} \widehat{T}_j(n) \tag{3.3}$$

Thus, we can define a confidence bound as a function of the reward obtained so far, which manages a trade-off between the exploration and exploitation of the search tree. Most of the Tree Policies studied so far in literature (and detailed in [37]) are applications of the Multi-Armed Bandit problem. Some of these are described below:

Random The nodes are selected at random from the parent, not depending on any reward estimate.

UCB-1 The child chosen in this case is the one that maximizes the value $\mu_j + C_P\sqrt{\frac{2\ln n}{N_j}}$, in which μ_j and N_j are respectively the average reward and the number of times the child was visited, and n is the number of selections that lead to the current node. The first term vouches for the exploitation of the previously visited choices with the highest reward values, and the second encourages the exploration of undiscovered nodes. For a better tuning of this trade-off, the exploration parameter C_P is introduced. The Monte Carlo Tree Search algorithm using this Tree Policy is usually referred to, in literature, as UCT (Upper Confidence Bounds for Trees).

UCB-Tuned Tuning of the bound on UCB-1 accounting for the variance. The equation that defines this upper bound is then $\mu_j + C_P\sqrt{\frac{2\ln n}{N_j}\min\{\frac{1}{4}, V_j(n_j)\}}$, where $\frac{1}{4}$ is an upper bound on the variance of a Bernoulli distribution and V_j is an upper bound on the the variance of the machine j.

ϵ-Greedy This policy requires the computation of $\epsilon_n = \min\{1, \frac{cK}{d^2 n}\}$, where K is the number of children, n is the child being considered, $c > 0$ and $0 < d < 1$. In this work, we consider an approximation of this formula, as found in [38]; the arm chosen maximizes the value of $\mu_j + \frac{\epsilon_n}{N_j}$. This calculation determines that the probability to play the node with the biggest reward value is equal to $1 - \epsilon_n$; following the same logic, a random node will be selected with probability ϵ_n. This is an adaptive solution, since the probabilities change with each updating of the node's rewards. The tuning of this policy is made with the choice of the greedy parameter ϵ.

The *Simulation Policy* has to deal with how the domain is played out in the simulations that are performed from the leaf node onwards, in order to produce a value estimate. In a regular Monte Carlo Tree Search, this is done simply by performing random moves from the ones that were not retrieved yet (*Default Policy*) and, afterwards, to update the statistics of each node that was traversed in that specific path.

However, there are many more variations related to this policy, such as the *Reflexive Monte-Carlo Search* ([39]) that uses a Meta-Monte-Carlo search to inform the simulation of the next timestep (level $n-1$ informs level n, and so on), starting at level 0 in which the policy is random, or the Nested Monte Carlo search ([22]), which is similar to the previous method but uses gradient ascent on the policy at each level of the nested search. Both these policies intend to emphasize actions taken by best solutions so far, helping push the simulations into their neighborhoods.

Types of Monte Carlo Tree Search Algorithms

There are several alternatives of application of the Monte Carlo Tree Search methods. They depend especially on whether or not the algorithm builds the entire tree, choosing the best node only after the budget is depleted or if a greedy choice is made, where at each timestep we get the best node from the current level of the tree we are developing, which is then considered to be the new root, forgetting everything that came before ([40]).

We can then define the two variations on the algorithm as:

- **Flat Monte Carlo**: A Monte Carlo method with uniform move selection and no tree growth.

- **MCTS**: A Monte Carlo method that builds a tree to inform its policy online.

For this problem, only the MCTS implementation will be considered. The Flat Monte Carlo approach was also tried, but the results were very poor, since in the case of interplanetary trajectories, the search tree is both limited in depth, due to pruning, and in breadth, due to the discretization of the continuous states. So, the tree does not possess enough depth for this implementation to yield a significant improvement on the results.

Therefore, this algorithm can be modified to suit the needs of our specific model. For that, we have to bear in mind that the previous detailed methods were designed with the aim of being applied to two-player games, such as Go - the problem tree has unlimited depth and breadth and when each move is played out, the entire tree that comes before is forgotten. Thus, it is also impossible to determine what is the overall best solution.

However, our search tree is limited by constraints, making it advisable to implement an extra step to the computation: the *Contraction* step, as described in [3]. This consists of removing the entire tree branch traversed at the end of the simulation (once it reaches a leaf), while keeping the statistics of the nodes and storing the best overall path. Thus, the tree search never repeats previously visited leaf states.

3.3.2 Rewards

A policy depends on a reward estimate for each node, which is calculated from the results of the simulations run from the node onwards. The reward value is bounded and scaled down in order to fit the interval [0 1], as indicated for this type of problem in [41], so the Tree policies can be applied. Having different goals, the problems considered in Section 2.3 have distinct equations for this computation.

Rosetta and Cassini Problems The reward value of the node for these cases is given by the following equation, taken from [3]:

$$X_j = \max\left(0, \frac{\Delta v_{max} - \Delta v_{tot}}{\Delta v_{max}}\right), X_j \in [0, 1] \tag{3.4}$$

in which X_j is the reward of node j, Δv_{max} is the maximum velocity imposed to the algorithm and Δv_{tot} is the total Δv of the analyzed path.

We can see that the reward X_j can either be 0 if Δv_{tot} is surpassed, or increasingly higher the smaller the final Δv is when reaching the goal.

As we can see in Section 3.3.1, the reward value for the node μ_j is considered an average value, as first determined on [37]. Thus, each time a node is updated, the reward X_j is averaged with the number of times the node was visited, yielding μ_j. Even so, both on [3] and [38], this value is considered not to be the average, but the maximum out of all the rewards obtained by the simulations performed so far. The reasoning behind this is that, as opposed to the work developed on [37], the latter ones are related to single-player games and, therefore, each X_j gives a guaranteed lower bound on the true node's value. Accordingly, this is the approach used on this project.

Asteroid Problem A simple linear regression is adequate for this model: the value of the reward is equal to 0 when only the starting asteroid is visited, and equal to 1 when all the 67 asteroids are traversed, according to the following equation:

$$X_j = \frac{1}{66} N_{visited} - \frac{1}{66}, X_j \in [0, 1], N_{visited} \in [1, 67] \tag{3.5}$$

in which $N_{visited}$ is the number of asteroids in one path.

3.3.3 Parameter Tuning

The variations of the MCTS algorithm that can be implemented, specifically the UCB-1 and the ϵ-Greedy policies, rely on the definition of parameters C_P and ϵ. Other variables have to be established with proper reasoning in how to do so, especially the ones related to the pruning of the search tree.

There is a total of five parameters to be defined: C_P, ϵ, maximum Δv, maximum mission duration and maximum number of swing-bys for a specific trajectory. The last two are easy to determine: for

the maximum mission duration, we chose the values 4000 and 3000 days for the Rosetta and Cassini missions respectively, as shown in Section 3.2; for the Asteroid problem, a maximum of 6 years (2191.5 days) of mission is established. The concept of maximum number of swing-bys only makes sense for the Cassini and Rosetta problems, but this limit is not imposed; although this decision makes the tree very big, there is no reason to constrain the spacecraft in this way from an engineering standpoint - the result with the lowest Δv may be one with many swing-bys, as long as the limit for the mission duration is not surpassed. The concept of maximum Δv is also not applicable to the Asteroid problem, since this value will not affect its reward in any way, and therefore will only be analyzed for the Rosetta and Cassini ones.

The methodologies implemented for each of these three parameters are described in the remaining of this Section.

Maximum Δv

The upper Δv limit for a given trajectory is a very important parameter in the algorithm for the Rosetta and Cassini problems, since it balances the pruning of the tree with the knowledge stored on the nodes on each simulation: after we reach a solution in the end of a simulation, the resulting Δv of the path is used to update the statistics of the other nodes in the way described in Section 3.3.2. In this way, a very high Δv limit may make the algorithm run for an infeasible amount of time, but a low one can make it too greedy. If the majority of nodes get a zero reward by surpassing the Δv threshold, there will not be enough valuable information to provide the tree on the Selection step.

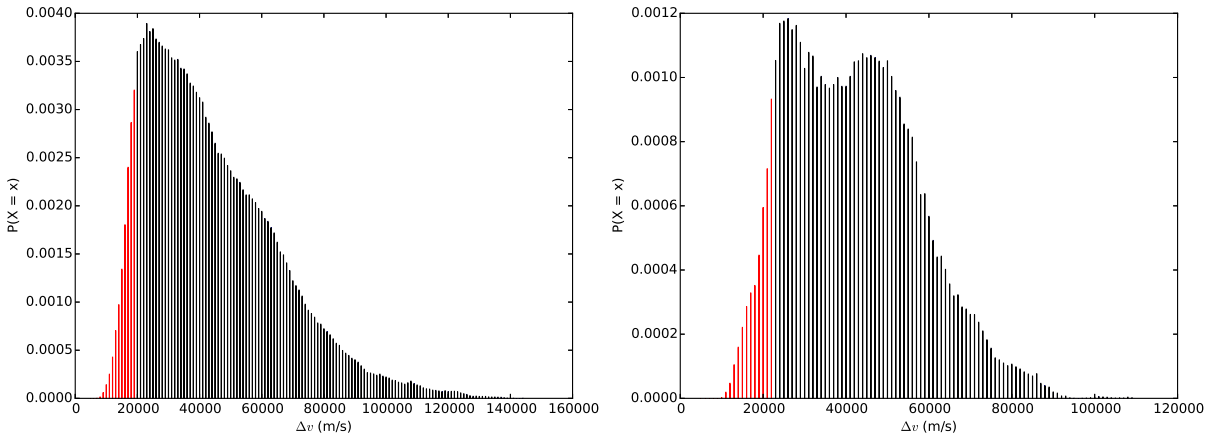
We want for the Δv threshold to allow for a significant number of nodes to contribute to the learning of the algorithm. We have defined this amount as 10% of the total visited nodes, for a proper balance between pruning and learning. Thus, the method to get this threshold is simply to run the algorithm on a significant budget and perform an histogram of the visited Δv at the end of each simulation. The results can be seen in Figures 3.3.

The red section on the histogram indicates the lowest 10% values of the final Δv of the solutions. For the Rosetta Problem, it is possible to determine that these fall just under the threshold of the 20 km/s mark; for the Cassini Problem, this happens on the 23 km/s value, which is consistent with the fact that its best solution, as fixed in Section 3.2.1, has a greater Δv than the Rosetta one. In this way, these are the values fixed as Δv thresholds.

ϵ and C_P

There are already some pre-defined and studied values for the parameter C_P: in particular, $1/\sqrt{2}$ is often utilized, as we can find on [37]. However, since we are dealing with a very different model and, most importantly, a single-player case, it would be best to determine the required parameters based on the algorithm's performance.

Rosetta and Cassini Problems In order to find the most adequate parameters, we have utilized different values of ϵ and C_P, inside the interval $[10^{-3}\ 10^{1}]$, on 4000 runs of the algorithm ([3]). In the end, we obtain a list of the Δv found for each parameter - the goal is to choose the parameters that



(a) Rosetta Problem

(b) Cassini Problem

Figure 3.3: Study of the Δv after each Simulation on a run of MCTS

appear to yield the best results. It is important to denote that, this being a stochastic algorithm, two runs with the same budget and the same parameter value may yield different final Δv, which is why the number of runs is significant for the study.

In order to make this claim valid, we wanted to test the variation of the parameters with changes in the pruning mechanisms of the algorithm: Δv threshold, mission time, number of swing-bys, initial date and budget in terms of number of simulations. In the end, we will find a feasible interval in which C_P and ϵ may lie. We do not want a single value, since this will be different when the constrains for each problem are changed - rather, we consider better to find a value that fits roughly in all the intervals obtained from different problem formulations.

These different implementations are shown on Tables 3.6 and 3.7, respectively for the Rosetta and Cassini problems. There are fourteen different cases, distinguished by a number from 1 to 14. Figures 3.5 and 3.6 represent the Δv distribution of the solutions taken from the parameter search for all the cases described on Table 3.6, related to the Rosetta Problem. For the Cassini Problem, the same is depicted on Figures 3.7 and 3.8, corresponding to the cases described on Table 3.7. The number of cases evaluated on the latter problem is smaller, since the effect of changing the same parameters on one problem is expected to have the same effect on the other, for they are similar in nature.

The parameter search gives us a series of points that relate a parameter value with the lowest Δv found in that run, for a given budget - in the figures, these points are colored in gray. This information has to be treated so that we can uncover the best parameter interval. We can, for instance, employ the method utilized on [3], to fit a Gaussian curve to the lowest Δv values (which are considered to be smaller than a given Δv threshold, regardless of the number of data points this gathers) and getting the mean of the Gaussian distribution as the best parameter. However, this method is not very consistent with the constraints now imposed on the problem and would yield results that do not correspond to the data that can be seen.

Being so, the method we created involves dividing the space into bins. Several sizes for these were tried, and the results that matched most accurately with the visual data use bins with 80 points each.

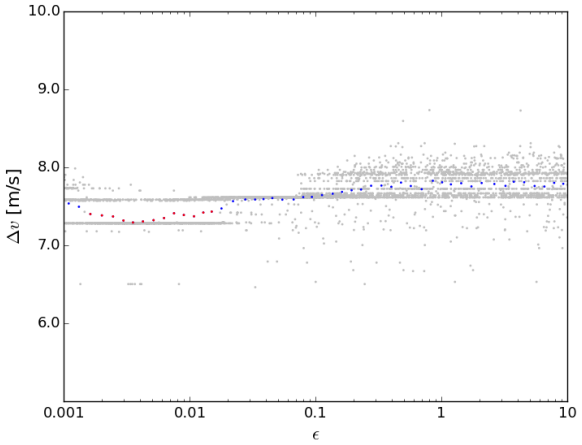
The average Δv of every bin was computed and is represented in blue. Afterwards, an adequate threshold of 150 m/s was applied to get the points that fall under this value, which are colored in red. These consist on the interval of adequate values for the parameter being tuned; the limits of this interval are shown on Tables 3.6 and 3.7 under labels *Min* and *Max*.

Number	Budget	Δv (km/s)	Time (days)	Flybys	Fixed	Parameter	Min	Max
1	50K	20	4000	×	Yes	ϵ	0.0029	0.0126
2	50K	20	4000	×	No	ϵ	0.0015	0.0098
3	100K	20	4000	×	Yes	ϵ	0.0032	0.0119
4	100K	20	4000	×	No	ϵ	0.0016	0.0103
5	200K	20	4000	×	No	ϵ	0.0036	0.0104
6	50K	5	4000	×	No	ϵ	0.0011	9.2317
7	50K	100	4000	×	No	ϵ	0.0011	0.0023
8	50K	20	4000	5	No	ϵ	0.0016	0.0090
9	50K	20	7305	×	No	ϵ	0.0023	0.0080
10	50K	20	1826	×	No	ϵ	0.0050	0.0218
11	50K	20	4000	×	Yes	C_P	1.6823	2.4507
12	50K	20	4000	×	No	C_P	1.3041	2.2137
13	100K	20	4000	×	Yes	C_P	2.1299	3.6468
14	100K	20	4000	×	No	C_P	1.7539	2.5077

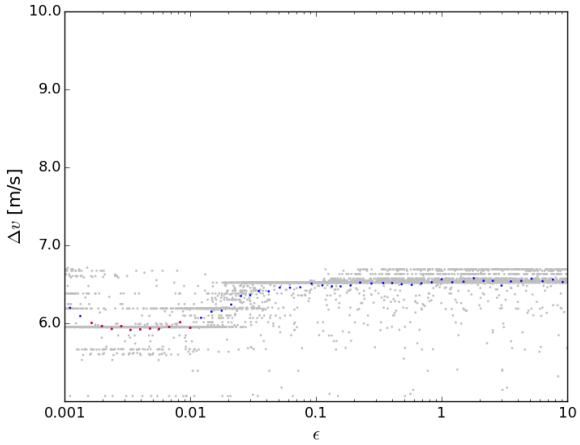
Table 3.6: Parameter Search for different constraints on the Rosetta Problem

Number	Budget	Δv (km/s)	Time (days)	Flybys	Fixed	Parameter	Min	Max
15	50K	23	3000	×	Yes	ϵ	0.0070	0.0152
16	50K	23	3000	×	No	ϵ	0.0040	0.0118
17	100K	23	3000	×	Yes	ϵ	0.0056	0.0172
18	100K	23	3000	×	No	ϵ	0.0054	0.0647
19	50K	23	3000	×	Yes	C_P	1.8073	3.0414
20	50K	23	3000	×	No	C_P	2.1127	3.57921
21	100K	23	3000	×	Yes	C_P	2.6205	3.7427
22	100K	23	3000	×	No	C_P	2.4893	6.2598

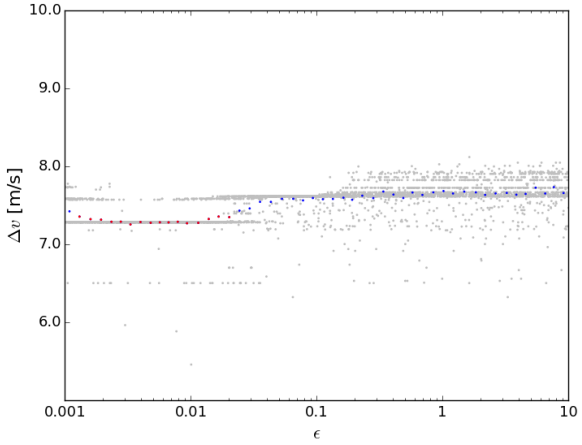
Table 3.7: Parameter Search for different constraints on the Cassini Problem



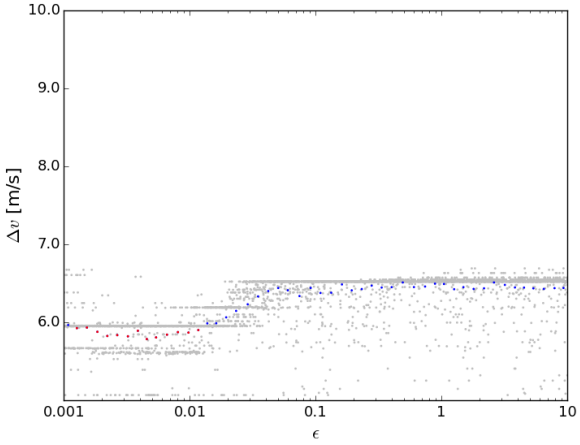
(a) Case 1



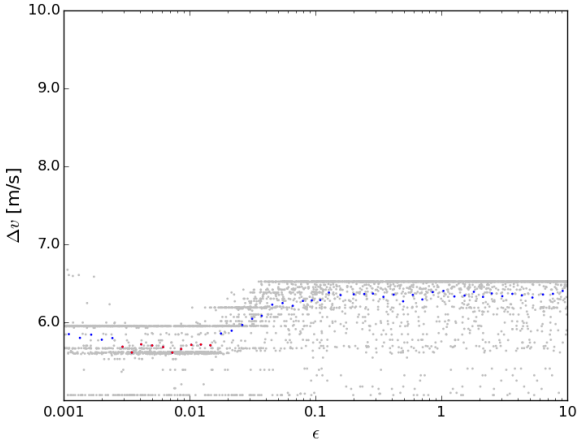
(b) Case 2



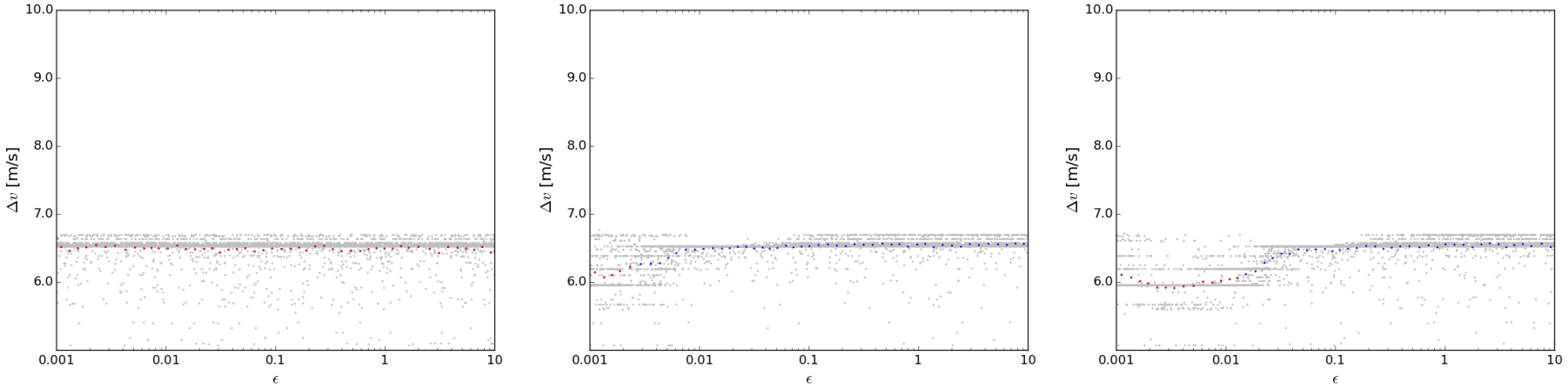
(c) Case 3



(d) Case 4



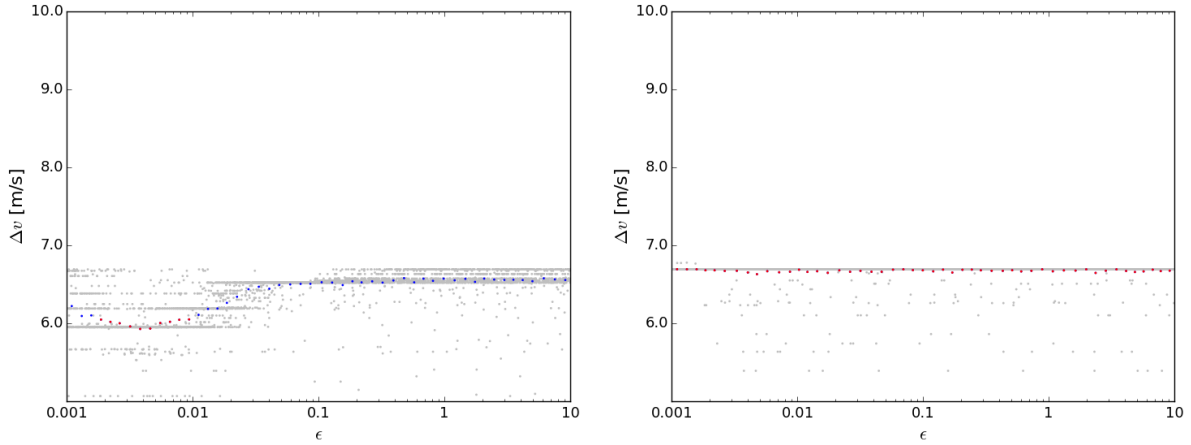
(e) Case 5



(a) Case 6

(b) Case 7

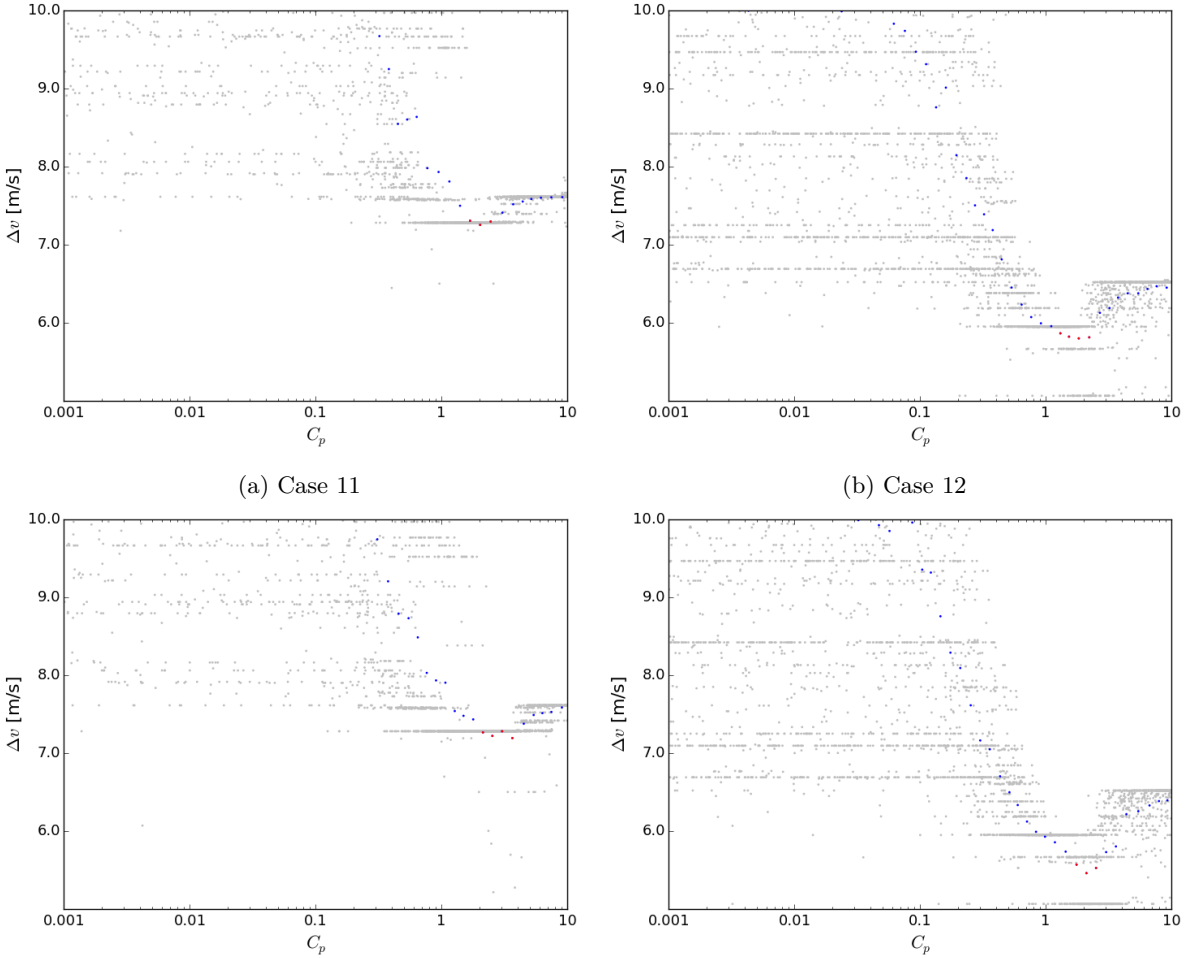
(c) Case 8



(d) Case 9

(e) Case 10

Figure 3.5: Distribution of Δv over different ϵ for the Rosetta Problem



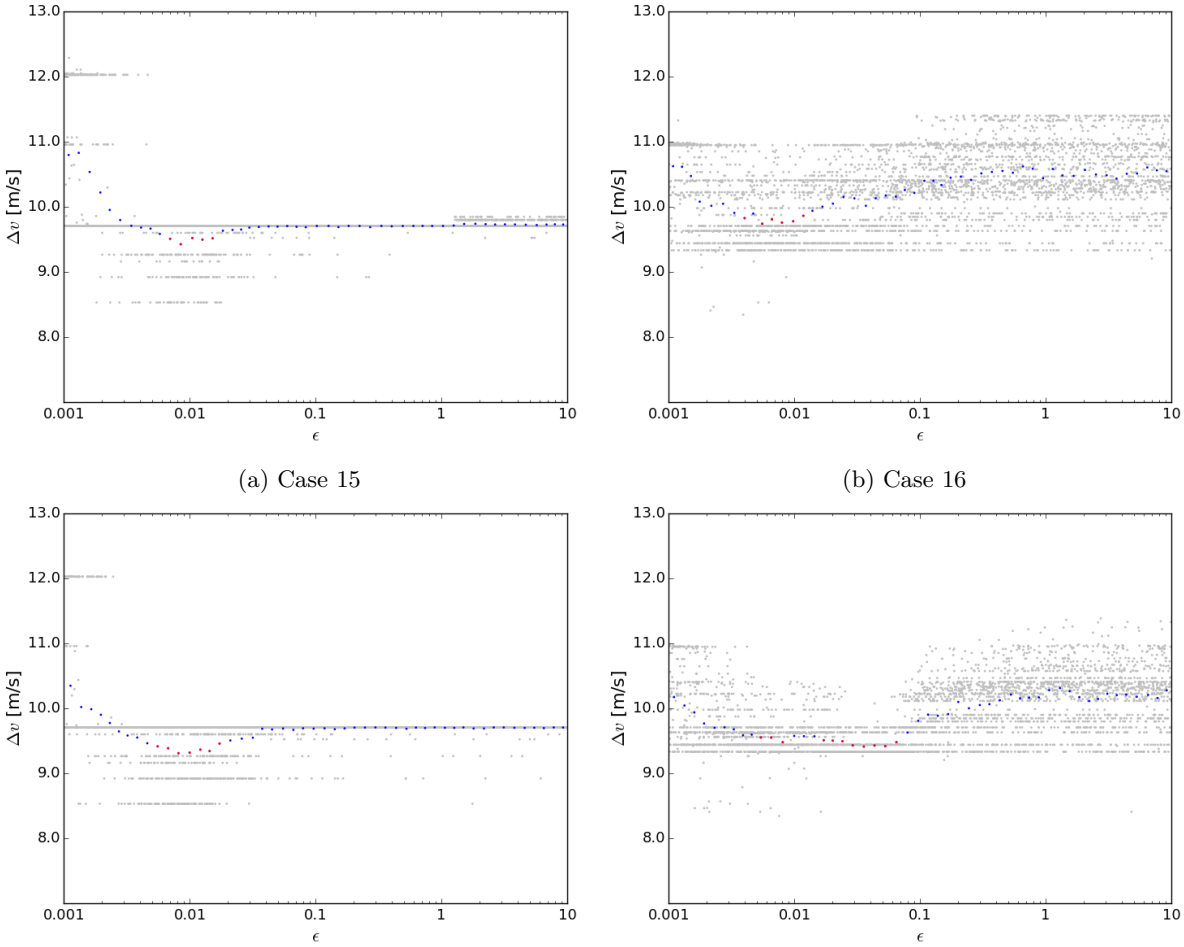
(a) Case 11

(b) Case 12

(c) Case 13

(d) Case 14

Figure 3.6: Distribution of Δv over different C_P for the Rosetta Problem



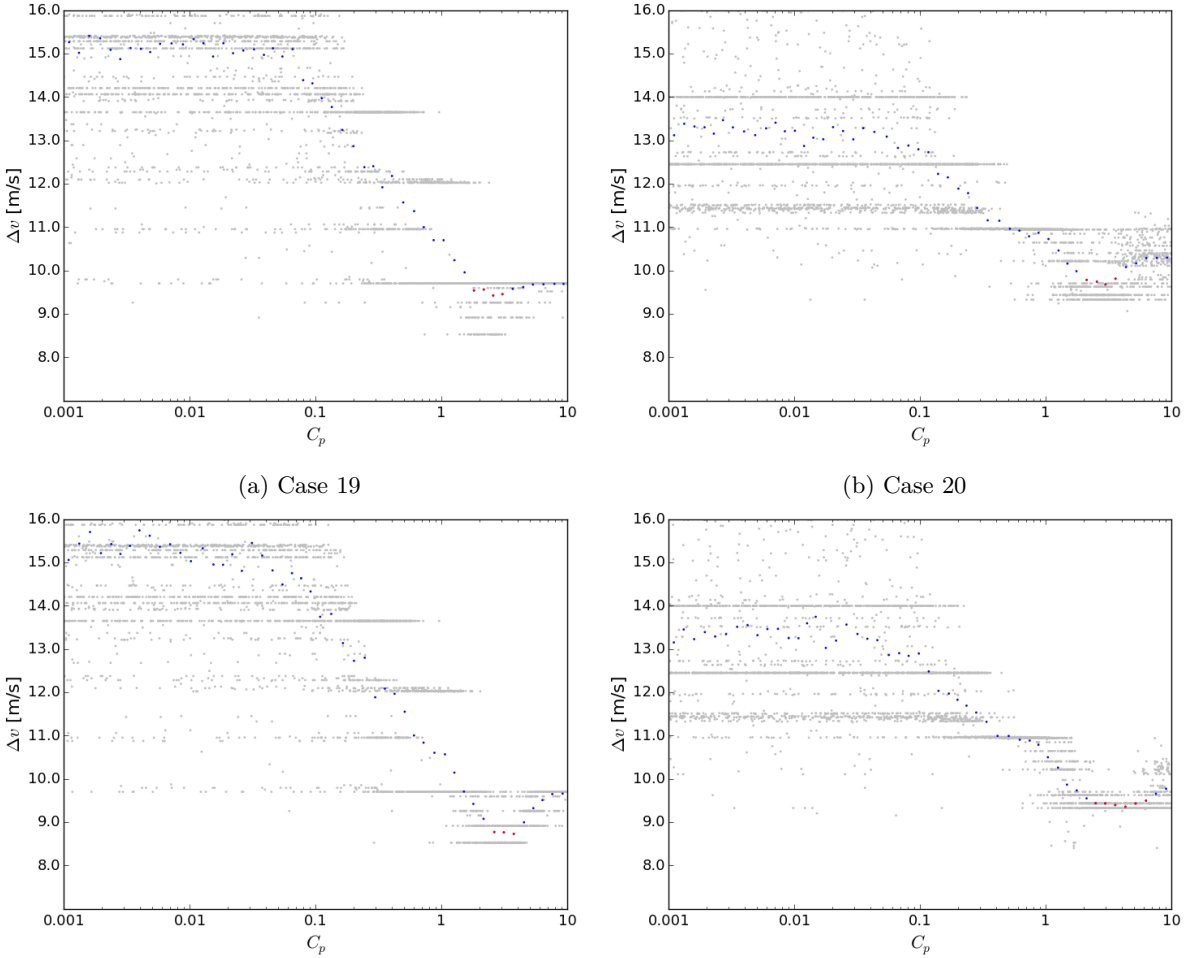
(a) Case 15

(b) Case 16

(c) Case 17

(d) Case 18

Figure 3.7: Distribution of Δv over different ϵ for the Cassini Problem



(a) Case 19

(b) Case 20

(c) Case 21

(d) Case 22

Figure 3.8: Distribution of Δv over different C_P for the Cassini Problem

For the Rosetta problem on Figures 3.5 and 3.6, the different cases give us perspective of the algorithm's behavior in response to changes in the defined parameters. We can see that the best solutions were generally found when the launch date was not fixed, although the state space is a lot bigger and the computation time increases. Also, the slope of the curve got steeper in this case, meaning there is a greater sensibility to the parameter used (due to the fact that the Selection step is repeated more often). Naturally, when increasing the budget from 50K simulations to 100K, the solutions found were also better; even more so when increasing the budget to 200K (Case 5), but this impact is not dramatic - it seems that increasing the budget by less than one order of magnitude does not make much of a difference.

On the contrary, constraining the problem too much (very low Δv or mission time, as in Cases 6 and 10) impacts the results massively. Solutions are found all over the parameter value range, which means the learning from the reward stops being done - once the Δv values are much dispersed, the algorithm is exhibiting a greedy behavior and more emphasis is given to the exploration as opposed to the exploitation of the search space. This happens because almost all resulting states from the simulations surpass the constraints. Even so, increasing both these parameters by a large value (Cases 7 and 9) does not actually give us better solutions than the ones found with the maximum Δv or mission time previously defined, which indicates that the thresholds are well established. Moreover, the swing-by sequence limit of 5 shown on Case 8 does not seem to have any impact on the final solutions found: most of them must possess this length already. However, the best solution found in Section 3.2.1 is bigger in length, so to impose this limit is still out of the question.

Using the intervals obtained from the Figures, detailed on Tables 3.6 and 3.7, we have determined the parameters to use as $\epsilon = 0.008$ and $C_P = 2.15$.

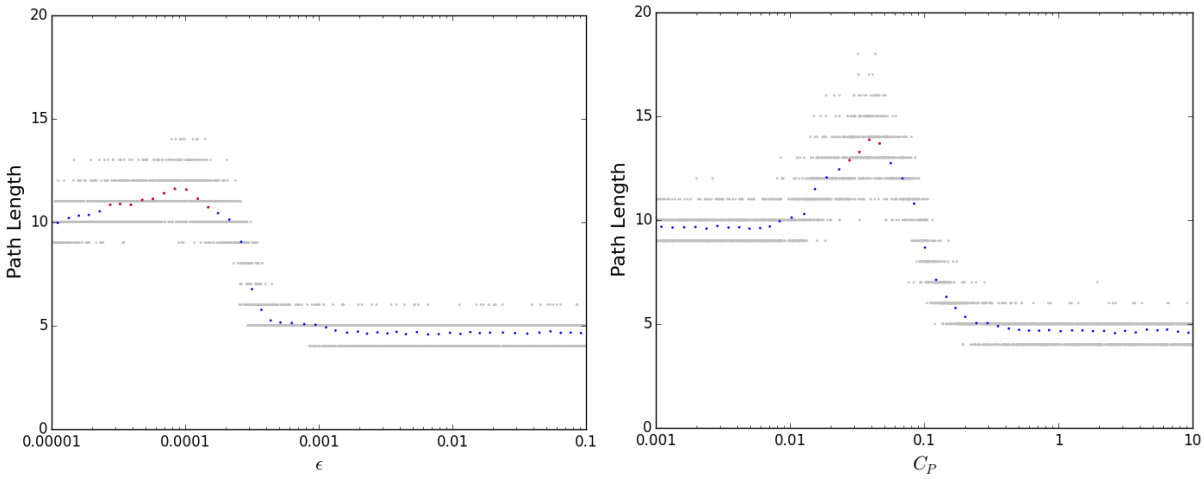
Asteroid Problem Considering this to be a different question mainly due to the objective function, it was sensible to perform a parameter tuning of the values of ϵ and C_P and expect it to yield different results.

The method for the analysis of the results is the one previously described, in which the algorithm is run 4000 times, with a budget of 100K simulations and the space in divided into bins of 80 points each. There was no point in trying to find an acceptable interval for this case, since the problem always poses the same constraints. The acceptable parameter interval, which consists of solutions differing from the best one by no more than one asteroid, is represented by the points in red that can be seen on Figures 3.9a and 3.9b. We can see that the best path lengths are found for much smaller values of the ϵ parameter than for the C_P, as it happened on the previous problems.

The feasible intervals are, for C_P, [2.77E-2, 4.63E-2] and for ϵ, [2.73E-5, 1.48E-4]. The values chosen for C_P and ϵ are of respectively 0.04 and 0.0001, which correspond roughly to the peaks in the figures.

3.3.4 Results

In this work, the Tree Policies to be tested are described in Section 3.3.1: the random one, UCB-1, UCB-1 Tuned and ϵ-greedy. We are interested in finding, not only the best in terms of final Δv, but also in terms



(a) Distribution of Path Length over different ϵ

(b) Distribution of Path Length over different C_P

Figure 3.9: Parameter Tuning for the Asteroid Problem

of computational time - this is a crucial characteristic to be taken into account, given that our search space is very big. The goal is to explore the tree and find a goal solution without taking as much time as an exhaustive search; with this in regard, two different budgets for the algorithm were considered: one in terms of the number of simulations performed and another in terms of computational time.

In order for the comparison to be meaningful, taking into account that the MCTS is a stochastic algorithm, each implementation of a different Tree Policy with the two budget types was run twenty times. The data is organized in the form of *boxplots*. These graphs show, in the vertical axis, the variable to be analyzed (the Δv or the path length); the horizontal red line inside the box symbolizes the *median* of the data set (the value in the set that divides it evenly in half); the lines that form the box are the *quartiles* (which divide the dataset in quarters). Inside the box are, then, 50% of the observed results, while the extremes are connected to the box on both sides via *whiskers* (vertical dashed lines). Any point ranging between 1.5 times the box's length below or above it is called an *outlier* and is represented as a cross.

In this way, the results from the Rosetta, Cassini and Asteroid Problems are shown below. The best solution of each problem, found by exhaustive search, is represented as a green horizontal line. All the launch dates were fixed for the value established in Section 3.2.1, for the state space becomes much smaller this way.

Rosetta and Cassini Problems The boxplots demonstrating the solutions are depicted in Figures 3.10a and 3.10b for Rosetta, and Figures 3.11a and 3.11b for Cassini.

We can see that the Random policy, on every test case, is obviously very bad, since the algorithm does not possess any information about the nodes that can be learned. The exploration of the search tree depends only on a random choice, and therefore there is no exploitation of promising areas of the tree (the reward does not influence the Selection process). In this way, there is also a great variability in the results, caused by the exploration of very distinct parts of the tree - considering the incredibly big state space, the randomness allows for a very broad exploration and improbable trapping on local

minima states. In this way, this policy may arrive to the best solution (as it happens on Figure 3.11b), but there is nothing to be inferred of that.

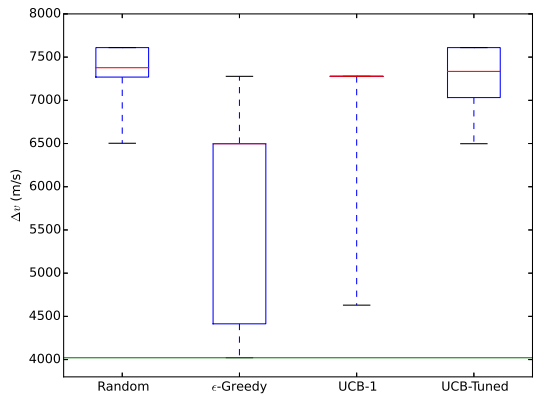
Moreover, the results for the budget of one hour are better than for the one million simulations - on the first case, the algorithm explored a significantly bigger part of the tree. For this policy, we confirm that the random choice made on the Selection phase is very cheap in terms of computational time - it is the least consuming out of all the policies.

However, on the case of the UCB-1, the results found are similar in both budgets: actually, this is the Tree policy that takes the longest to evaluate the nodes with the confidence bound, which makes it the most computationally expensive. Besides, the exploration range is relatively narrow: due to the time constraint, it seems to get trapped in a local minimum very often, showing the exploitative nature of the method. It is, however, better than the Random policy, since it uses the reward to visit the best states it can.

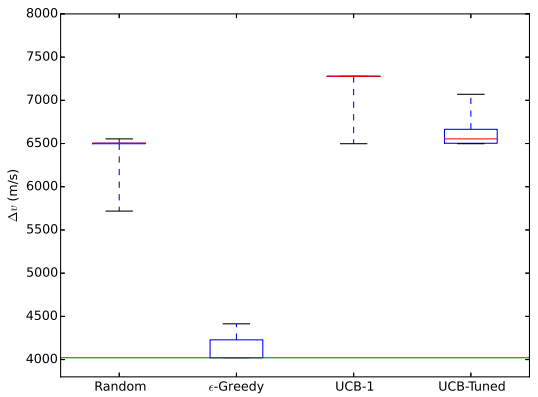
The UCB-Tuned is very similar to this method in terms of behavior. It does not take as long to run as the UCB-1 policy, but its results are not necessarily better: we can see that the Tuned version outperforms the regular one on the Rosetta case, but not on the Cassini one. This may happen due to a multitude of reasons: first of all, on the Rosetta case, the UCB-1 policy seems to get trapped in a local minimum very often, something that happens for the Tuned version in Cassini, deteriorating both performances. It is not clear, however, which one of these policies is better on this specific domain. In [37], it is observed that the UCB-Tuned policy generally outperforms the UCB-1 but it is not possible to prove a regret bound (difference between the optimal reward and the one obtained by the Tree policy). Even so, both results are clearly outperformed by the ϵ-Greedy policy.

In terms of computational cost, the ϵ-Greedy policy is the lightest, after the Random one. It is the one that yields the best trade-off between optimal results and computational time, when finely tuned. It is also important to denote that this finds the best solution for both problems many times within the budget of one hour, while the exhaustive search took approximately 300 times longer - the median of the plots is the optimal solution on the Rosetta problem with the budget of one hour (Figure 3.10b) and on the Cassini problem on both budgets (Figures 3.11a and 3.11b).

We can also distinguish some outliers in all of the plots. This happens due to two reasons. First of all, the nature of the boxplot itself considers as outliers points that are distant to the central 50% in the way previously explained. So, if the solutions are often trapped in local minima, the times where this does not happen can be considered outliers; this is frequent on the Random policy, where the variability of the solutions is great (Figures 3.10a and 3.10b). The other possible scenario, that does not seem to happen here, is when the computer is at the limit of its RAM usage; in this situation, the results with a time budget may suffer due to the computational overload.

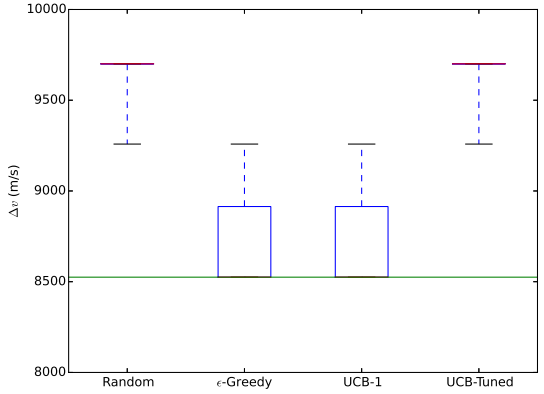


(a) Budget of 1M Simulations

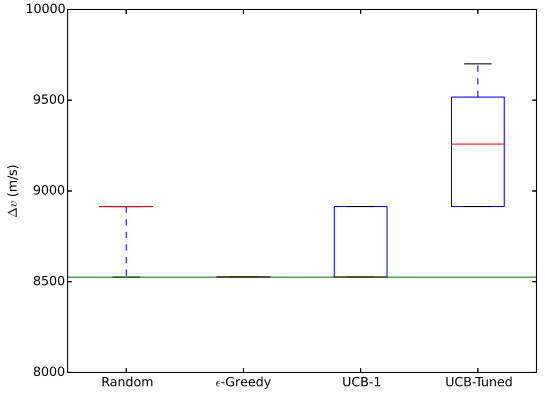


(b) Budget of 1 Hour

Figure 3.10: Different Tree Policies for the Rosetta Problem



(a) Budget of 1M Simulations



(b) Budget of 1 Hour

Figure 3.11: Different Tree Policies for the Cassini Problem

Asteroid Problem As we have considered in Section 2.3.3, there are two pruning mechanisms to this code: surpassing the mission time or the maximum permitted mass, m^*.

In Section 3.1.1, we have considered three possible time discretizations for this problem. We have decided also to compare the four different policies on the Regular grid, expecting the ϵ-Greedy to be better, to check if it performs just as well on a problem that employs a different evaluation function.

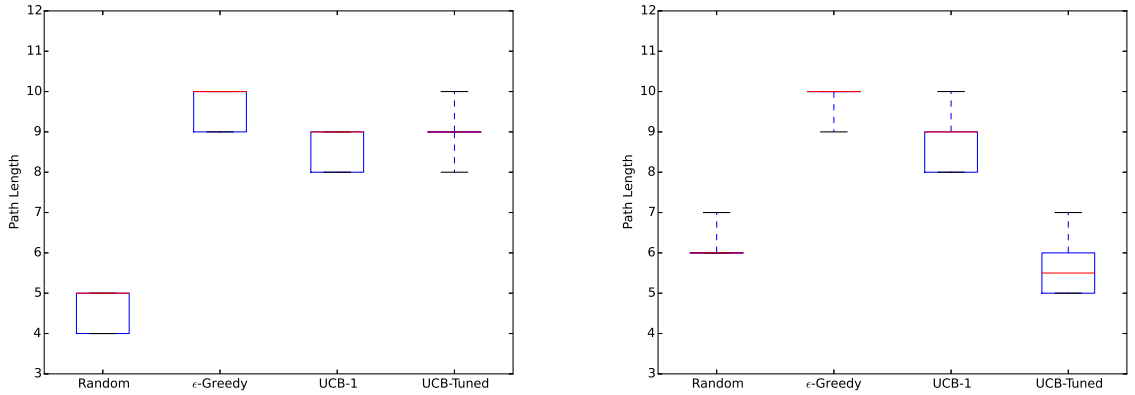
Observing Figures 3.12a and 3.12b, we see that the ϵ-Greedy policy proves again to be better than the others. A close second is the UCB-1 policy, which arrives to solutions whose length is equal to the best ones found by the previous policy. Clearly, the reward computation proves to be very important in this case, since the Random policy yields the worst results. The path lengths found so far range from 4 to 10, still far away from our goal of 14.

To do a comparison between the time discretizations, the ϵ-Greedy policy with $\epsilon = 0.0001$ was used, since it remains the policy that presents the best results and explores the most of the search tree in the least time.

We can see that the range of path lengths is very small, making the results vary significantly with the addition of only one asteroid. Moreover, we see that the Logarithmic time grid clear separates itself from the others, achieving at best, with the same Tree policy, paths composed of 12 asteroids. This comes from the fact that the time intervals are not equally spaced but, instead, are more concentrated on a range of the time interval for which more maneuvers are allowed.

The results for the Forced and Regular grids are similar, the Regular being lightly better on the simulations budget and the opposite happening on the time budget. Although the idea behind the Forced discretization is promising, the code ends up testing too many times of flight, which takes time from an efficient exploration of the possible asteroids. Furthermore, the times of flight in the solutions are at the higher range of the allowed interval, especially in the beginning of the trip, when the spacecraft is at its heaviest and cannot travel fast within the mass constraints. Being so, the algorithm spends too long testing times of flight values that are infeasible to fully explore the possibilities. To produce good results, it should run for a longer time period.

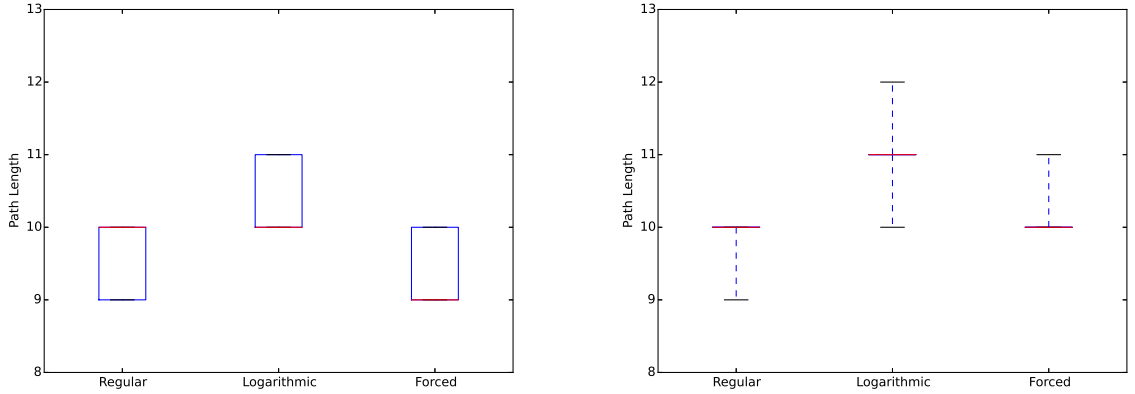
It is important to denote that all these solutions have terminated in the path length observed due to time constraints, and not because the spacecraft's mass exceeds the maximum allowed. In this way, the time discretization proves to be a determinant factor on the final path length. An efficient utilization of departing and arrival periods may allow time for the visit of more asteroids, making the consideration of times of flight as continuous to be a possible benefit to this problem.



(a) Budget of 1M Simulations

(b) Budget of 1 Hour

Figure 3.12: Different Tree Policies for the Asteroid Problem



(a) Budget of 1M Simulations

(b) Budget of 1 Hour

Figure 3.13: Different Time Discretizations for the Asteroid Problem, using the ϵ-Greedy Policy

Chapter 4

Hybrid Optimization

4.1 Introduction

Considering the importance posed by the discretization of the time intervals, a different approach to deal with the problem proposed in this project is not to use a grid but, instead, to consider the times of flight continuous. In this case, an exhaustive search of the state spaces is impossible and, therefore, the employment of the previously considered techniques is infeasible. So, we propose two different approaches for the hybrid optimization problem, named after its duality in nature: continuous times of flight, discrete planets for swing-bys. One is a recent variation of the UCT (MCTS with UCB-1 as Tree Policy), called **HOOT** [42], a search algorithm that develops the continuous states into a binary tree using confidence bounds and searches for the best solution within a budget. The other is to apply to this case an **optimization algorithm** integrated with the MCTS and several of its variations, which will be described in the following sections.

4.1.1 Software

During this Chapter, PyKEP will still be used to model the interplanetary maneuvers, but other Python libraries will also be employed. The most notable one is the Optimization package on Scipy, the Scientific Python library. From this, we will use convex optimization functions (SLSQP, L-BFGS-B, COBYLA and TNC), and global ones, namely the Differential Evolution function. We will also employ cProfile, a Python script profiler.

4.2 HOOT

The HOOT algorithm (Hierarchical Optimistic Optimization applied to Trees) is a fairly recent addition to the Monte Carlo Tree Search family of algorithms, as described in [42]. It eliminates the need to discretize the action space by developing the continuous time states into a binary tree and producing simulation reward values for these nodes.

The HOOT algorithm has two distinct behaviors depending on the nature of the node being selected:

if it is discrete (planet node), the regular MCTS is used; when it is continuous, the HOO algorithm ([43]) is employed for the Selection phase. The HOO consists in starting with the time window in question, taking its middle point as a root and expanding a binary tree from there on, decomposing the continuous space. At each time step, when queried for an action to take, the algorithm chooses the child node with the biggest B-value until it reaches a leaf, when it is subdivided in two different nodes, and this process is repeated at each time step for action selection. This routine can be seen on Figure 4.1; it can be more or less throughout in the exploration of the continuous space depending on the time we want to spend on the HOO part in relation to the entire MCTS.

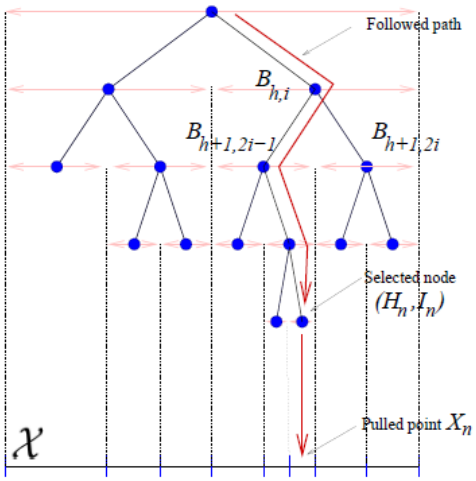


Figure 4.1: Binary Tree and node Selection for a continuous set of nodes using the HOO algorithm [43]

The B-value is a high probability confidence upper bound on the mean value for each arm, computed from the statistics of the simulations made from that node forward on each previous time step. The B-value is formulated in terms of the upper confidence bound $U_{h,i}$ in the way of Equation 4.1:

$$U_{h,i}(n) = \mu_{h,i}(n) + \sqrt{\frac{2\ln n}{N_{h,i}(n)}} + \delta \tag{4.1}$$

in which $\mu_{h,i}(n)$ is the reward estimate for the node, defined as the average of the results obtained throughout the simulations, δ is the width of the estimation (always bigger than zero, and ideally contracting while the budget is depleting) and $N_{h,i}(n)$ is the number of times node i at depth h has been visited until that point. For nodes that have not yet been sampled, $\hat{R}_{h,i}(n) = U_{h,i} = \infty$.

4.2.1 Application to the Problem

The interplanetary trajectory design search, as previously seen, is a hybrid problem - the discrete part is the option to choose between the planets, and the continuous one is the time interval regarding the times of flight. Thus, the HOOT cannot be applied in its regular way, since the final solution is not only a value inside a continuous interval: it is a sequence of these values, interleaved with the planet choices for each swing-by.

Therefore, the application of the HOO algorithm for this specific tree is not obvious. The main difference from the MCTS algorithm is revealed in the Expansion step: if the next node is to be a planet, then the algorithm expands all possible planets and chooses randomly between them. However, if the next node requires a choice of time, the HOO is applied as described in Algorithm 2.

```
If Node is Time then
    While budget is not over
        Selection
            node ← node.Select(B-value)
            B_{h,i}(n) = min{U_{h,i}(n), max{B_{h+1,2i-1}(n), B_{h+1,2i}(n)}};
        end
        Expansion
            node ← node.Expand(Binary)
        end
        Simulation
            state ← state.Move(Random)
            If Terminal then
                break;
        end
        Backtrack
            node.Statistics ← Reward
        end
    end
return
```

Algorithm 2: HOO

The algorithm works integrated in a Monte Carlo Tree Search framework. In order to figure out the values for each node and expand the binary tree, random simulations have to be carried out. After the first binary expansion and reward estimates update, the selection becomes possible. This cycle continues until the binary tree reaches a certain depth level, depending on the budget. In the end, we return the best N nodes to the MCTS algorithm in the expansion phase. In this way, if during the MCTS we find the next node to be a planet, we expand all of the possible ones; if it is a time node, we expand as many as the HOO algorithm allows.

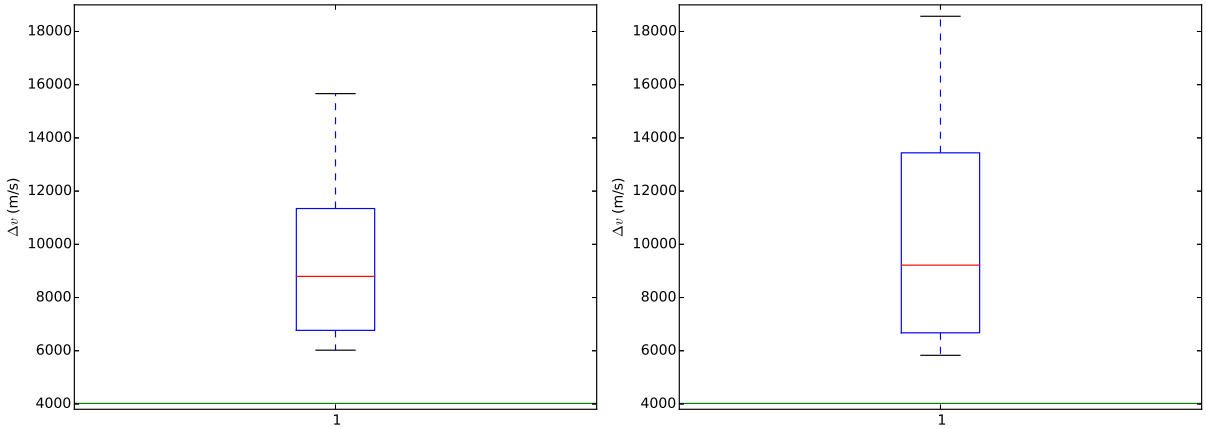
As we can see in Algorithm 2, the number of nodes the HOO part of this method returns depends of the specifications given to it, which are the budget given to the HOO algorithm in terms of expansion of the binary tree, and the number of time nodes that this delivers to the entire MCTS. When there is a planet expansion, S more nodes are added to the tree (S being the number of possible swing-by planets). Considering this, we have implemented three different variations of the HOOT algorithm: one that returns the single best node, one that yields the best five (the same as the number of possible swing-by planets in the Rosetta mission) and another that delivers all nodes whose reward is bigger than zero (or, when this is not possible, the also the top five). We have decided to use the second implementation, since to return one node only would be too few and the other variation is very similar to the chosen one.

For these experiments, we have found that a budget of 250 simulations for the HOO to be an adequate choice, which proved to be a good trade-off between the MCTS simulations and the level of expansion of the binary tree.

4.2.2 Results

Considering the expansion of the binary tree, the algorithm will take much longer to run for the same budget in simulations. Therefore, it is not expected for this method to arrive to solutions of the same quality as the ones described on Chapter 3 and, also, to the the optimal solution found previously in Section 3.2.1, since the tof values are not discretized in that form. Even so, it is useless to perform the exhaustive search on this tree due to the fact that it is not finite.

Following the same method employed in Section 3.3.4, 20 runs of the algorithm were performed on the Rosetta problem for budgets of one million simulations and one hour. The parameter used for the Tree Policy is the one determined on Section 3.3.3, $C_P = 2.15$. The green line shown on Figures 4.2 represents the best solution for the Rosetta problem defined in Section 3.2.1.



(a) Budget of 1M Simulations

(b) Budget of 1 Hour

Figure 4.2: Results of HOOT on the Rosetta Problem

We can see that the results given by one million simulations are slightly better than the one hour, behaving the same way as the UCB-1 policy on the Rosetta problem. It is observable that the median of both solutions is one. The worst solutions, in this case, reach values of the order of 18 km/s, something unthinkable of for a space mission. Thus, this method is not a good one for the Hybrid Optimization problem.

4.3 Optimization

An optimization algorithm has the purpose of finding the best solution regarding certain criteria and constraints. It is applied to an objective function and, mathematically, consists on its minimization, considering additional equations, inequations and bounds for the variables in question. In this way, a generic optimization algorithm can be described by the following statements:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, n \\ & h_j(x) = 0, \quad j = 1, \ldots, m \end{aligned}$$

in which f is the mathematical function to optimize, x is the design variable, $g_i(x)$ are the n inequality constraints and $h_j(x)$ the m equality constraints to the problem. This formulation has to be adapted for the test cases in question: either the Rosetta and Cassini problems, or the Asteroid one.

Rosetta and Cassini Problems

Previously, we have considered the state space as a finite tree that possessed two different types of nodes: times and planets. The depth levels of the tree alternate between these two, allowing for the calculation of a Δv for each Lambert leg, as described in Section 2.2.1. Naturally, this formulation changes with the consideration of a hybrid problem.

In this kind of implementation, we no longer can use the time grid previously imposed. Instead, the time of flight becomes the parameter to determine - the design variable. So, our optimization function should get a sequence of planets and deliver a string of times of flight that corresponds to the optimum Δv. We could do so each time we reached a time node (which is when the Δv calculation is done) and, at the end, to sum all the Δv of the entire sequence, as in the discrete case. However, it is not guaranteed that the minimum Δv will be achieved, since to constrain the times for only one Lambert leg may impair the optimization of the full sequence, as it is imposing a certain position for the spacecraft at the determined time. In this way, it is logical to optimize the times of flight only after we have a full trajectory.

The optimization problem can be formulated like this:

$$\begin{aligned} \underset{t}{\text{minimize}} \quad & LambertPath(t, p) \\ \text{subject to} \quad & L_i \leq t_i \leq U_i, \quad i = 1, \ldots, len(p) \\ & p = predefined \end{aligned}$$

in which the design variable is an array of times of flight (t) and the objective function is the $LambertPath$, which takes the times and the predefined swing-by sequence p to deliver the optimum Δv. The design variable t falls inside the interval $[L_i\ U_i]$, the window when the trip is allowed.

Using this approach, this problem becomes a non-convex, multi-variable bounded optimization (due to the fact that there are launch and arrival time windows that have to be respected for the trajectory to be feasible) that does not include equality or inequality constraints on the design variables. This demands a powerful and robust optimization algorithm that does not require the function's Hessian matrix, which is too hard to derive.

Asteroid Problem

The Asteroid problem cannot be as easily converted to a continuous optimization one as the previously discussed ones. In these, the full sequence is developed and, afterwards, the times of flight are returned for the smallest Δv. In this case, we cannot list the full sequence beforehand: even with the time constraints, the search tree would be too big. For instance, let us consider a sequence whose times of flight in between the asteroids were all the minimum allowed - its length would be of 36 asteroids. Since there are 67

available asteroids including the fixed first one, the amount of possible sequences we get from this is of the order of 10^{60} - the state space is huge. Moreover, we have no guarantee of the feasibility of the sequence regarding the mass constraints.

Being that the only limitations to the problem are mission duration and the spacecraft mass and knowing that the path length is impossible to optimize by itself, there are two design variables that seem adequate for optimization: the mass threshold for termination (m^*) and the spacecraft's own mass.

If we use m^* as our design variable, the algorithm's purpose would be to maximize it (the same as minimizing its negative), since a greater m^* is a bigger threshold for the spacecraft's initial mass, which will give us a wider search tree of possible sequences to explore. However, to maximize m^* is the same as doing so for the time of flight - testing, all the values yielded corresponded to the upper limit to our bound, 330 days. This is confirmed by our model, which uses a method for the computation of low-thrust trajectories implemented by Landau ([30]). From Equation 2.9, we can see that, when a gets smaller, this increases m^*, and to diminish the first variable, the time of flight must be increased. In this way, m^* as a design variable is not a viable approach to our project.

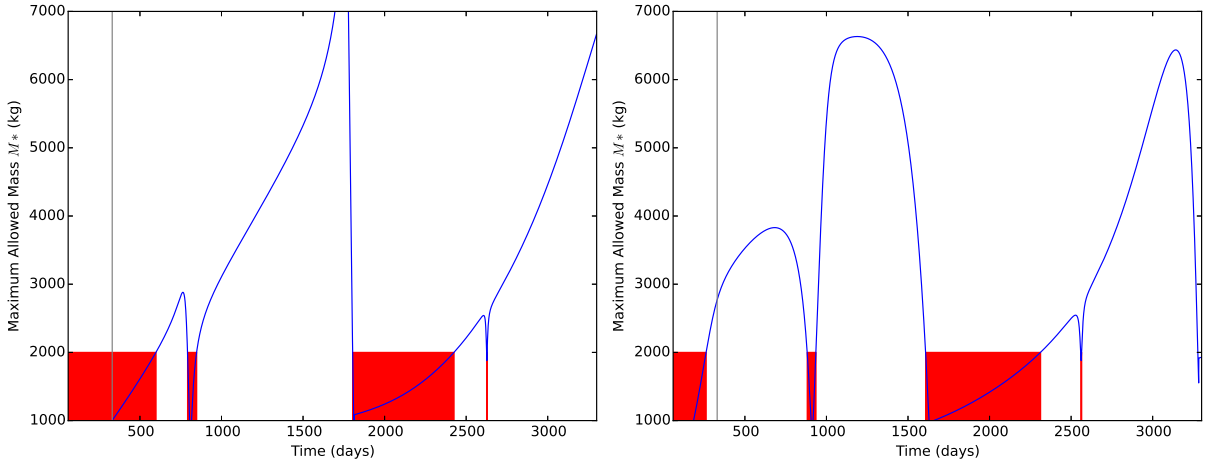
Being so, the design variable that seems the most adequate for optimization is the vehicle's mass after each maneuver between asteroids, since a smaller mass will give more maneuverability to the ship. The goal is then to find the time of flight that yields a trajectory which leaves the ship as light as possible and, by doing so, allows us to explore more trajectories that would otherwise be infeasible in the discrete case, instead of terminating too soon. Theoretically, the paths found should only be limited by the mission duration.

This is not an ideal strategy: considering the time of flight bounds, it is possible that the lowest mass is very close to the upper limit of the interval, which leaves us less room to perform maneuvers, since the remaining allowed time is shorter, while there can be other feasible values for masses that correspond to smaller times of flight. In order to check these possibilities, to study the behavior of a spacecraft's mass over time would be useful.

Thus, we have plotted the maximum allowed mass for each maneuver of the spacecraft over a time frame of 60 to 3330 days, from the fixed starting asteroid 658 (Mimosa) to asteroids 523 (Aletta) and 932 (Lujiaxi). The launch date is 8395 MJD2000 and the mass of our spacecraft is 2000 kg. Thus, whenever m^* is below this value, the maneuver is infeasible: the time slots corresponding to these parts are represented in red. It is important to denote that, in our problem, the upper limit for the time of flight is of only 330 days: therefore, this value is marked with a line of the color gray. In this way, we can study the behavior of the function over a large time frame, but also discern it in the scope of our problem.

As we can see on Figure 4.3, for the asteroid 523, as opposed to 932, there is no possibility of travel on the slot from 60 to 330 days - some paths are infeasible regardless of the time of flight. Furthermore, the graphs show a series of convex slopes opening on the bottom; if this behavior is consistent through most of the asteroid paths and epochs, the minimum mass will also correspond to the earliest allowed time of flight, a reasonable assumption for the quality of this method.

The optimization problem, in this case, is formulated as follows:



(a) To Asteroid 523

(b) To Asteroid 932

Figure 4.3: Maximum allowed mass for traveling from Asteroid 658 from a 60 to 3300 days time of flight window

$$\begin{aligned} \underset{t}{\text{minimize}} \quad & AllowedPath(t, p) \\ \text{subject to} \quad & 70 \leq t \leq 330 \\ & p = predefined \end{aligned}$$

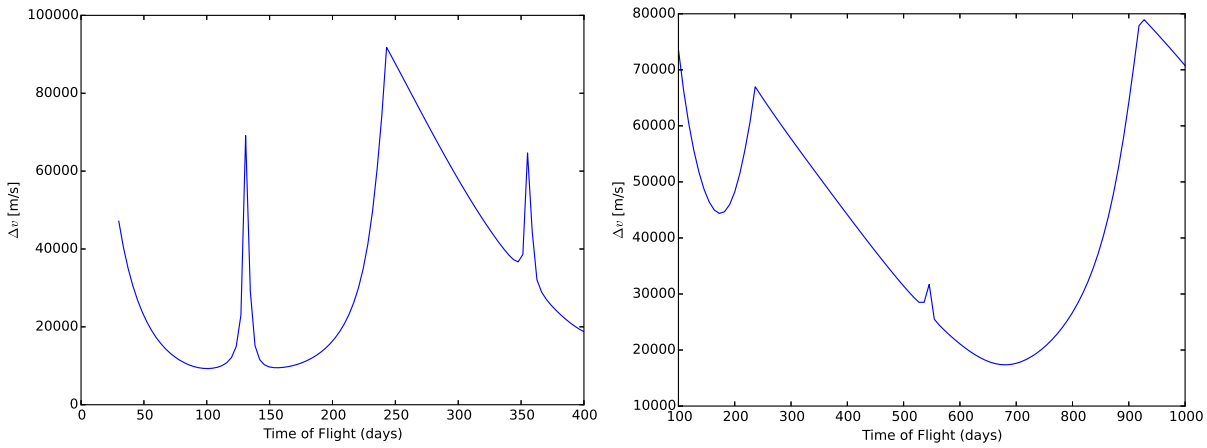
in which the design variable is the time of flight (t) and the objective function is the $AllowedPath$, which takes the t and a predefined path p composed of only two asteroids at a time to deliver the optimal spacecraft mass. This function also returns an infeasible value for the algorithm to recognize in the case of non compliance to the constraints posed. This problem is bounded and does not include equality or inequality constraints on the design variables.

4.3.1 Optimization Techniques

Optimization can be done in two ways: *local optimization* and *global optimization*, depending on the characteristics of the function to be minimized. If the latter is *convex*, it has only one minimum; if it is *non-convex*, there can be several **local** minima that can 'deceive' the algorithm into considering it the **global** one, the desired result. We can then observe that the complexity of the optimization is much higher on the second case, since there are as many candidates for the solution as there are local minima, as opposed to just one.

Thus, it is important to check, for the Cassini and Rosetta test cases, whether the function computing the Δv using planets and times of flight ($LambertPath$) is convex. We can observe Figure 4.4, which displays the Δv over the times of flight inside the intervals allowed as launch window, in a swing-by from Earth to Venus and another from Earth to Mars.

As we can see, for both these situations, the $LambertPath$ function is non-convex. Although it would be too much to analyze all the possible combinations of bodies and times of flight, we cam extrapolate the absence of a global minimum to most of them, since it is consistent with the fact that the Δv depends on



(a) Earth to Venus

(b) Earth to Mars

Figure 4.4: Function of Δv over time for the Trajectory between two Planets, departing time at 1574 MJD2000

the epochs of these planets and planetary motion is not a linear subject: at a time, two celestial objects can be at much more distant points in their orbits than at the previous time step.

Given the nature of this problem, we can approach it in two ways: to implement a global strategy, which is necessarily more costly in computational terms but fits the problem characteristics; or we can apply a local method using an informed initial guess, which will find the local minimum closer to that point. For the first case, we have considered the *Differential Evolution* method to be the best, since it is the only global optimization method available in the Scipy library that does not require an initial guess and can encompass bounds for the design variable. For the second, we have considered as candidates all the the algorithms that can support bounded, multivariate problems: the SLSQP (Sequential Least Squares Quadratic Programming), the L-BFGS-B (Limited-Memory Bounded Broyden–Fletcher–Goldfarb–Shanno), the TNC (Truncated Newton Conjugate-Gradient) and the COBYLA (Constrained Optimization BY Linear Approximation). The last one, although not bounded, has the possibility of including inequality constraints that fulfill the same purpose.

Global Optimization - Differential Evolution Differential Evolution is an optimization method that falls in the realm of Evolutionary Algorithms, a relatively new branch of multidisciplinary global optimization [44].

Similarly to the Darwinian theories on species evolution, Evolutionary Algorithms start with an *initial population* (selection of a string from the design variables in the allowed state space), that will 'compete' to pass on their genes (characteristics) to the next *generation*. The selection of individuals that get to do so is done by the *fitness evaluation* of them - the variables (individuals) that yield the smallest values in the objective function have the best fitness. However, the genes are not directly passed on to the next generation; they may suffer some kind of *mutation* or *recombination* (meaning they may be changed or recombined with other design variables).

The entire process can be summarized by the flowchart in Figure 4.5. We start by selecting individuals from an initial population, by fitness; they reproduce by one of the methods considered, originating the

children, which are candidates for the next generation. This new population is the consequent pool of selection candidates, and this goes on until a termination condition is reached.

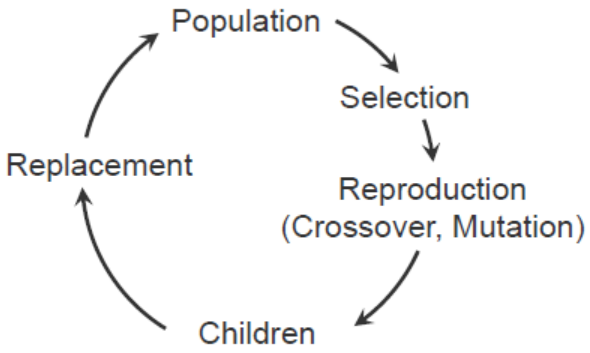


Figure 4.5: General Flowchart of an Evolutionary Algorithm

Being a specific algorithm in the realm of Evolutionary techniques, Differential Evolution consists on a stochastic method applied to multivariate functions which works in the following manner: we start with a sampled population from the entire state space, whose size is predetermined. This sample can be taken with methods like the *Latin Hypercube* ([45]) or simply by random, but should cover the entire state space. The remaining process is common to every Evolutionary Algorithm: the distinction between different methods comes from the mutation and recombination parameters. In the case of Differential Evolution, the candidate solutions are mutated by mixing with other individuals: this is done by adding the weighted difference between two population vectors to a third vector, as in the following equation:

$$b' = b_0 + F(pop_0 - pop_1), F \in [0, 2] \tag{4.2}$$

in which b_0 is the best set of design variables, F is a mutation parameter that controls the solution's variability and pop_0 and pop_1 are two sampled vectors from the population. The method of *crossover* is used afterwards and consists on increasing the diversity of the mutant vector by mixing it with the parameters of the target vector (computed as determined on [46]), to yield the trial vector for the next generation. The selection is done between this trial vector and the target: the one that yields the lowest cost function value is the fittest.

This algorithm is very robust, although mathematically simple. It has been compared to several established non-convex optimization algorithms, delivering better performance [46]. These kinds of methods are very good for noisy functions with no available gradients, but the optimality of the solution is not guaranteed: several parameters have to be carefully tuned for the global optimum to be achieved with a greater probability, such as the number of function evaluations, the mutation and recombination weights and the population size.

This optimization technique is integrated with the MCTS algorithm in Section 4.3.2.

Local Optimization As previously mentioned, the L-BFGS-B, the SLSQP, the TNC and the COBYLA are the algorithms considered for the informed optimization. They differ from the Differential Evolution one in the way that they are not stochastic but, instead, rely on an initial guess of the optimum. The first three utilize a gradient descent until zero, where a minimum is found. In opposition, the last one works without computing any gradient but, instead, by linearizing the actual problem.

The BFGS (Broyden–Fletcher–Goldfarb–Shanno) algorithm encompasses techniques by all of these scientists that named it; it is a quasi-Newton method in the sense that it does not require a direct evaluation of the Hessian matrix, saving computational effort. It uses line searches along a search direction subject to the bounds imposed. The method is an iterative one; it starts with the computation of the Cauchy point (a local minimum), the definition of a direction for the line search, computation of the gradient and update of the quadratic model for optimization (detailed algorithm in [47]). The L-BFGS-B is an adaptation of the BFGS with a memory limit and encompassing box constraints. It is an easy to use method, with low iteration cost, indicated for cases where information about the Hessian of the function is unknown.

The second method, the SLSQP, is also quasi-Newton and can include hard constraints such as equations and inequations, as described in [48]. Therefore, it is more costly in computational terms. So, theoretically, the L-BFGS-B algorithm will perform better that the SLSQP, since it takes less time to run and the quality of our solutions depends on an efficient exploration of the search tree.

The TNC is adapted from the Nonlinear Conjugate Gradient (NCG) method to encompass bounds on the design variables. It is a Newton method, so it tends to converge in more iterations than the quasi-Newton ones (such as the L-BFGS-B and the SLSQP). The algorithm is explained in more detail in [49].

The last algorithm, the COBYLA, is applied to functions whose gradient is unknown. It works by approximating the actual constrained optimization problem with linear programming problems. Once these are solved, the quality of the solution is evaluated relatively to the original constraints. This is done until a point where the step size has to be diminished for the solution's quality to increase, as detailed in [50].

In order to figure out the optimizer to utilize, we have tested the four of them on the Rosetta and Cassini problems with an ϵ-Greedy Policy. The SLSQP algorithm posed convergence problems in several of the runs and is, therefore, excluded as a possibility. The COBYLA algorithm is the one that performs the worst of all. We have concluded the L-BFGS-B to be the algorithm to use, since it yields the smallest values in Δv for the same budget out of all these methods that were tried. The quality of the L-BFGS-B is also verified for other domains in literature ([51]).

This method is integrated with the MCTS algorithm in Section 4.3.2.

4.3.2 Implementation

The implementations described here will reflect the solutions for the Rosetta and Cassini test cases. Since the Asteroid problem is very different in terms of optimization, it will be regarded in the end of this section.

We have defined that, to utilize an optimization function, we must first predetermine the swing-by path sequence, since the optimization is only applied to the times of flight. Thus, a simple way to get the optimum Δv for the mission would be to define all the possible planetary sequences with a depth-first search algorithm and, afterwards, to apply the optimizer (given the launch windows between planets) to get the times of flight. Since these are added only after the sequences are constructed, the new search tree has no information about the Δv or total mission duration of its swing-by sequences. Therefore, it is not possible to prune nodes that reach a maximum Δv or surpass a mission time, as it was done on the combinatorial problem. The only way we can impose a time constraint is to limit the sum of the lower bounds of the launch windows from each swing-by planet. In this way, this sum must be smaller than the maximum mission duration, and no planet can be added to the sequence if it surpasses this constraint. This can be represented by the following equation:

$$T_{mission} > \sum_{i=1}^{n} lb_i, \tag{4.3}$$

in which n is the number of planets belonging to the sequence, $T_{mission}$ is the maximum mission time and lb_i is the lower limit in the launch window to that planet i.

Naturally, one can see that, after running the optimization, some trajectories will be infeasible if the returned times of flight yield a mission time greater than the established maximum. Thus, after the construction of the tree of planets, the optimization is done for each branch and, once the times of flight and Δv have been delivered, the overall solution is pruned again to eliminate trajectories that do not obey the constraints. Yet, even after posing this bound, the size of the search tree is too big for the computation to be tractable, its construction too slow for the optimization performed in this way to be possible. This happens mainly due to the fact that the launch windows between some planets, for instance Earth and Venus, are very small: some trajectories can become very long from using almost only these planets in sequence.

This can be solved in one of three ways: for the first one (**Constrained Length Sequences**), we add a limit to the length of each sequence. The latter was chosen to be equal to the size of the best path in the combinatorial domain: seven planets in total for Rosetta and six for Cassini. With these bounds, we can generate the possible sequences (373 in total for Rosetta, 56 for Cassini) and get the best ten of them in terms of Δv for comparison with the algorithms already applied. Since we have no possibility to make an initial estimate of the time, there was no doubt in using the Differential Evolution algorithm in this case. It is important to highlight that this method can only be used as a benchmark to compare the combinatorial sequences to and analyze the possibilities that the hybrid optimization presents us, for the problem changes dramatically from the initially proposed one.

The second approach, which does not require any pruning, is to integrate this on a MCTS framework, in which the sequences are evaluated and their quality backpropagated to the tree (**MCTS with Initial Guess for Local Minimization** and **MCTS with Global Minimization**). This can be either done with the regular MCTS solution as an initial guess for the optimization or, as an alternative, it is possible

to utilize the global optimizer only on the planetary sequence, without dwelling on the times of flight. This approach is more adequate for comparison with the Combinatorial Problem, using similar Tree Policies and time and simulation budgets.

The third approach (**Best Tree Sequences**) is to consider the best solutions of the discretized space, to apply to them the optimization methods and check how they compare with the full integration of the MCTS. In this way, we can study whether the optimization of all the sequences is feasible, or if it is better to do so only for the best found by the MCTS algorithm as implemented on Chapter 3.

The results yielded by all of these possible approaches are detailed on Section 4.3.3. Following, we describe how the integration of the MCTS with the optimization techniques, the second approach, was implemented.

MCTS with Initial Guess for Local Minimization

We have integrated the L-BFGS-B method with the MCTS algorithm in the following manner: first, using the MCTS as described on Chapter 3, we get a full sequence of planets and times of flight from the Earth to the target planet. Then, the local optimization method is applied, using the times of the determined sequence as an initial guess of the function's minimum Δv. After this, the reward computed using this Δv is backpropagated to the tree, following the regular MCTS algorithm sequence.

The suspicion is that the discretization used, described in Section 3.1.1, is fine enough so that, in between two points of the same grid, there is no more than one minimum and, therefore, the optimized solution is always better than the discrete one. In this way, instead of searching through the whole interval of times of flight for the best result, we indicate to the algorithm where a good solution already is.

By this logic, the solutions found should be at least of the same quality as the discrete ones, if not better. However, this is not so trivial. If the budget applied to both algorithms was in terms of simulations, the hybrid optimization version would always find the same or best solution. However, in terms of time, we have to take into consideration that the explored space will be smaller, since the optimization is run for each solution found by the already time-consuming MCTS algorithm. Also, the bigger the sequence, the longer the time to optimize it.

MCTS with Global Minimization

This method consists on the integration of the Differential Evolution algorithm for optimization of the times of flight for each sequence with the MCTS method. As opposed to the previous algorithm, on this case, the search tree consists only of planet nodes; the times of flight are added posteriorly, by employing the optimizer.

When developing the tree, once a termination condition is reached (which means we have arrived to the target body or surpassed the maximum mission time in the way indicated in Equation 4.3), the optimization of the full sequence is made and the nodes are updated with the determined times of flight and Δv. With those values of Δv, the reward is computed in the same way as detailed in Section 3.3.2, giving information to the tree about the quality of the path.

However, in spite of being computed in the same manner, the rewards on the combinatorial optimization and the Initial Guess cases influence the algorithm in a greater way than in this one. This happens because the former implementations give information both on the quality of time of flight and planet nodes, while during this one only planet nodes are being used - the information propagated to the tree is not as complete.

The computation of the reward is also to be considered: it is done in the same way as in Chapter 3 for comparative purposes, although the pruning of the Δv does not happen while the tree is being built.

Before we could employ the Differential Evolution algorithm, some more parameters needed tuning: these were the *popsize* (number of points composing the initial sample), the *mutation* and *recombination* values (by default, equal to 15, (0.5, 1) and 0.7, respectively). It is known that, if we increase *popsize*, *mutation* and decrease *recombination*, we are improving the algorithm's probability of reaching convergence ([46]). However, we will also get a higher computational time, since we are dealing with more sample points and mutation operations. What we must then determine is whether the convergence gains by modifying these parameters are impaired by the longer time spent by the algorithm on the optimization or not, considering the full run to be on a time budget.

In order for the comparison with other optimization techniques to be more meaningful, we have decided to test, for the *popsize* parameter, the value 169, which is the average number of times the MCTS with Initial Guess runs the optimizer on a given sequence - on this algorithm, the same trajectory is tested many times, with different times of flight as initial assumptions. So, we have decided to apply the global optimization to a pre-determined trajectory with different parameters: from the Earth to comet 67P, with swing-bys in Venus and Jupiter. We have variated the *mutation* values from (0, 0.1) to (1.8, 1.9), utilizing all possible combinations in between, and we have also done so for the *recombination* factor, changing it from 1 to 0. The final Δv corresponding to each unique parameter combination was the result of averaging 20 runs of the algorithm. For each value of *popsize*, this yielded 2090 different values, from which we can observe some conclusions.

The Δv given by the pre-defined values of the optimization function is equal to 5461.57 m/s. For a *popsize* value of 169, this becomes $\Delta v = 5438.03$ m/s. Most of the other results presented a variation in Δv of at most 100 m/s, which is not very meaningful - the best overall solution was $\Delta v = 5437.61$ m/s, for the values of mutation and recombination of (0.2, 0.4) and 0.1, respectively. We have found this variation not to be significant and, accordingly, decided to keep the function's pre-defined values - while still comparing the different assumptions for the *popsize* parameter.

Asteroid Problem

Not all the strategies applied to the previous problems can be implemented on the Asteroid one, due to the differences in its formulation. As observed in Section 4.3, the only way to use the times of flight as the optimization's design variable for the spacecraft's mass minimization is to consider only two bodies at a time. Using these together with the launch date from the first asteroid, the algorithm will be left with only one variable to optimize: the arrival time to the second one. This will be done each time a new asteroid is added to the sequence, until the constraints (mass and mission duration) make it infeasible.

In this way, the method applied here is analogous to MCTS with Global Optimization, which proved to be the only one with the potential to present quality results, since the use of algorithms requiring an initial guess would not guarantee a throughout search of the allowed times of flight. For instance, if we were to use as a hypothesis a time whose surroundings are all values that make the maneuver infeasible, the optimizer would be trapped in these states (the mass values which are not supported by the constraints are all interpreted by the algorithm as a value greater than 2000 kg).

4.3.3 Results

On this Section, we only use a time budget, since the algorithms are so computationally expensive that the budget in terms of simulations took too long to run, making it inadequate for our purposes.

Constrained Length Sequences

The best ten results that fitted in the time constraints are presented in Tables 4.1 and 4.2, respectively for the Rosetta and Cassini problems. All the times of flight are rounded to two decimal places, although the optimizer has the precision of thirteen (since the times are in days, this difference in precision is noteworthy). It is important to mention that all the final Δv found by the optimization process were confirmed to be correct and the maneuvers are physically possible.

The sequences are here shown just for information on their variability. We can see that most of the swing-bys are performed on Earth and Venus, Jupiter being too far to be a good option more often. On the Rosetta case, only one solution seems to be better than the one found by exhaustive search, which leads us to believe this constraint in number of swing-by sequences is not a very good approach to the problem.

T_0	Sequence	Times of Flight	Δv (m/s)
1574.14	E-V-V-E-J-E-67P	150.32, 442.49, 58.43, 714.74, 661.40, 1547.66	3233.27
1574.14	E-V-V-E-J-V-67P	150.20, 442.40, 58.35, 660.11, 626.03, 1542.02	4467.72
1574.14	E-V-V-E-E-67P	151.40, 443.35, 59.61, 639.06, 2088.28	4610.75
1574.14	E-E-M-E-V-E-67P	713.24, 605.33, 171.27, 284.67, 258.13, 1577.74	5176.55
1574.14	E-E-E-M-V-E-67P	693.55, 693.67, 481.65, 131.11, 387.46, 1423.64	5187.31
1574.14	E-V-V-E-E-M-67P	153.84, 444.15, 64.68, 665.33, 554.28, 1771.21	5395.97
1574.14	E-V-V-E-J-M-67P	150.26, 442.45, 58.39, 660.09, 534.51, 1136.36	5410.22
1574.14	E-V-V-E-J-67P	150.24, 442.44, 58.38, 675.81, 1355.17	5445.03
1574.14	E-V-V-E-E-E-67P	153.40, 445.79, 69.35, 685.56, 685.57, 1665.45	5463.06
1574.14	E-E-E-M-67P	698.27, 698.37, 490.20, 1752.00	6228.97

Table 4.1: Best 10 Sequences for the Rosetta case after Optimization, with Limited Path Length of 7

The best trajectory, the only one that improves on the combinatorial solution, is represented on Figure 4.6. It is possible to distinguish the planets depicted as colored balls, and their orbits around the Sun as lines of the same color. In this way, the Earth is represented with a dark blue tone, Venus in magenta, Jupiter in yellow and comet 67P in black. For the orbital maneuvers in between them, the color cyan is used. The plane represented is the ecliptic of J2000.

The length of most of the ten best Cassini sequences is smaller than the Rosetta ones because the

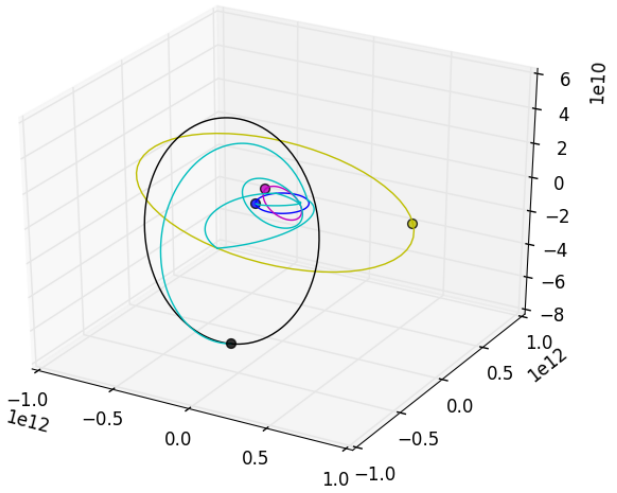


Figure 4.6: Best Constrained Length Rosetta Sequence

maximum time of flight is more constrained and Saturn is in an orbit farther away from the Earth's than comet 67P (Saturn's perihelion is at a distance of 9 Astronomical Units and the comet's is at 1.3).

T_0	Sequence	Times of Flight	Δv (m/s)
1574.14	E-V-V-E-J-S	195.71, 402.84, 53.66, 615.19, 2495.59	5698.92
1574.14	E-V-J-S	143.64, 752.22, 1719.94	7639.33
1574.14	E-V-V-E-S	192.70, 413.35, 52.96, 1972.00	8091.25
1574.14	E-V-V-J-S	197.04, 416.02, 609.73, 1978.93	9053.03
1574.14	E-V-V-E-E-S	177.14, 442.55, 69.81, 670.76, 1939.56	10782.07
1574.14	E-V-V-M-V-S	192.31, 449.40, 551.89, 102.49, 1957.37	11099.87
1574.14	E-E-E-S	691.02, 688.68, 1754.62	11605.79
1574.14	E-E-E-E-S	715.50, 713.92, 713.88, 1748.59	12263.56
1574.14	E-V-E-E-E-S	103.61, 258.91, 730.51, 674.57, 1857.78	12736.50
1574.14	E-E-S	611.38, 2500.00	12960.02

Table 4.2: Best 10 Sequences for the Cassini case after Optimization, with Limited Path Length of 6

On the Cassini case, there are three solutions that represent improvements on the combinatorial solution by exhaustive search. The best one is represented in Figure 4.7: the colors are the same as the ones used for the Rosetta problem, and Saturn (the end destination) is represented in purple.

It is important to mention the Differential Evolution method does not always find the global minimum, getting instead trapped in a local one, something that causes slight variations in the solution value. As such, some sequences may have yielded a Δv higher than the global minimum. In this way, these results are used as proof of the potentialities of this method, with possibility of finding even better trajectories.

MCTS with Initial Guess for Local Minimization

The solutions found are represented in the following figures, only for the ϵ-Greedy Policy, which was considered to be the best one in this problem. Since the model is similar to the combinatorial one, the

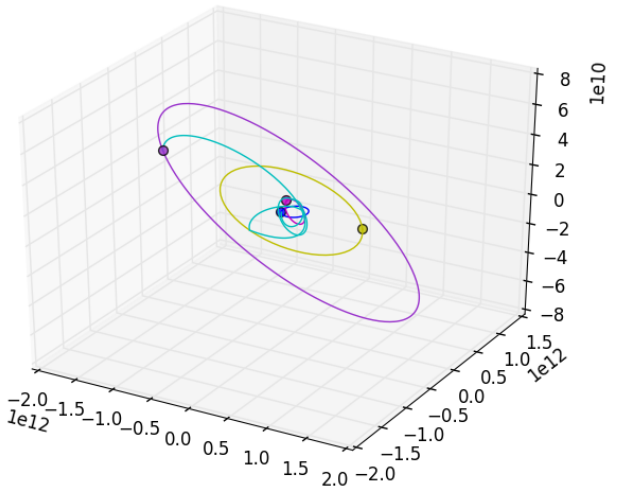
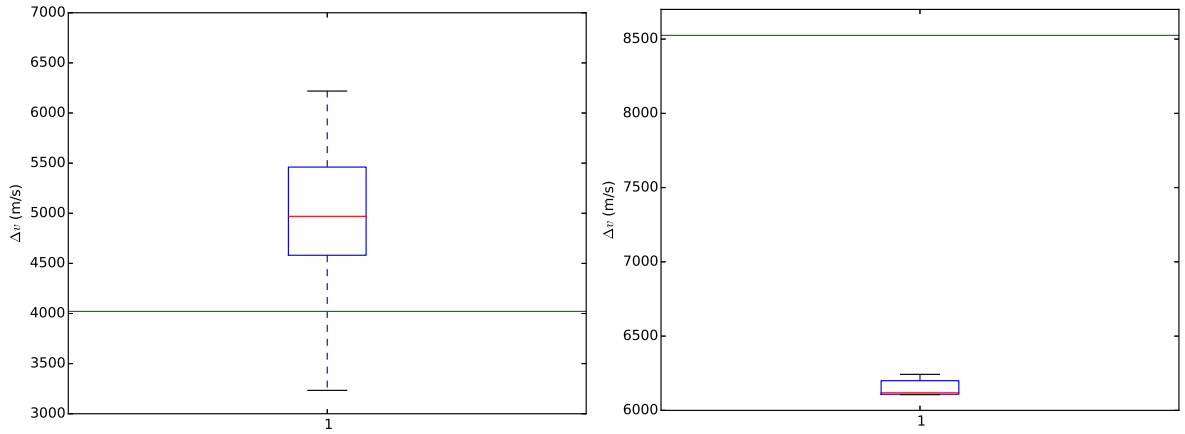


Figure 4.7: Best Constrained Length Cassini Sequence

parameters computed for the ϵ-Greedy policy are also employed here:



(a) Rosetta Problem

(b) Cassini Problem

Figure 4.8: Final Δv Results using an ϵ-Greedy policy for the Optimization with an Initial Guess with budget of 1 Hour

We can see that only the solutions for the Cassini problem are better on this case than on the combinatorial one, using the same budget. They are much less dispersed in terms of Δv and yield impressively better results, the best saving around $2km/s$ - which indicates us that the Cassini problem is the most affected one by the discretization made in Chapter 3. However, on the Rosetta case, some of the Δv are smaller than the combinatorial solution, but overall this method does not perform very well, since the optimization part of the algorithm weights on the performance, consuming computational resources. Accordingly, this option is not viable when considering this limited a budget.

In order to verify this claim, we have studied the algorithm with a Python profiler, a set of statistics that describe exhaustively the performance of all the parts of the program. In this way, we can observe

the running times of every function and routine and infer the ones that spend the most of the total computational budget. The profiler's report indicates a duration of 3600.72 seconds for the entire program (as expected, with a budget of one hour) and indicates that the function that took the longest to run is the minimization one (with the method L-BFGS-B), spending 98.996% of the total time to run. This is consumed in two different ways: the function to minimize, which receives the trajectory and various tentative time of flight sequences, computing the Δv for each of them by doing several heavy calculations (ephemeris of each planets at the given times, Lambert propagations, etc.); and also the optimization function on itself, employing the Limited Memory Bounded BFGS algorithm and several other subroutines. The function to minimize takes 65.130% of the total time, leaving 33.866% purely for this Scipy method.

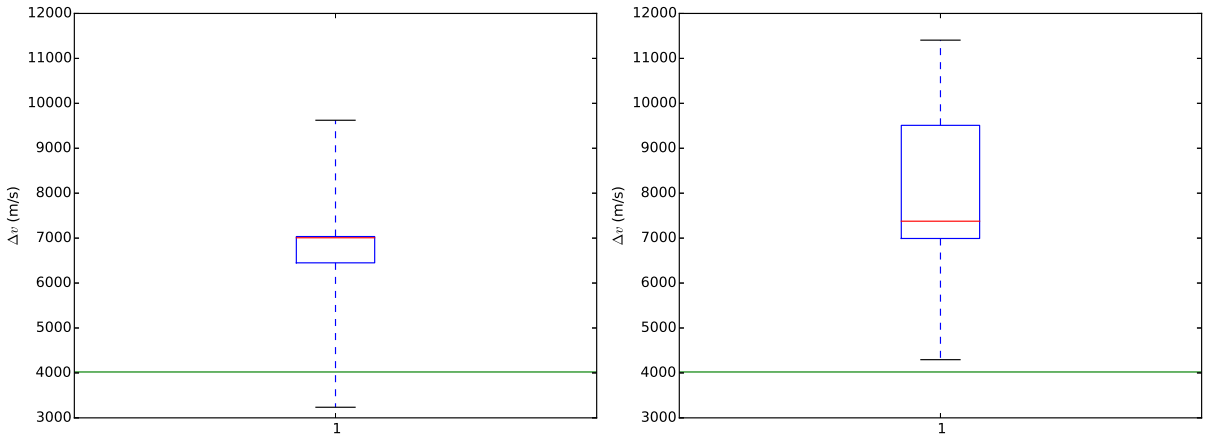
On the other hand, on the combinatorial case, the code was optimized in performance for all the ephemeris to be pre-computed and, for each sequence, only one Δv had to be calculated; in the hybrid problem, several Δv are computed for the same sequence in order to find the minimum. In this way, it is easy to infer that the quality in results gained by the throughout exploration of the continuous time states is impaired by the limited amount of time left to the exploration of other promising planetary sequences.

MCTS with Global Minimization

We have also chosen to use here the ϵ-Greedy policy, which was the one that proved to perform better.

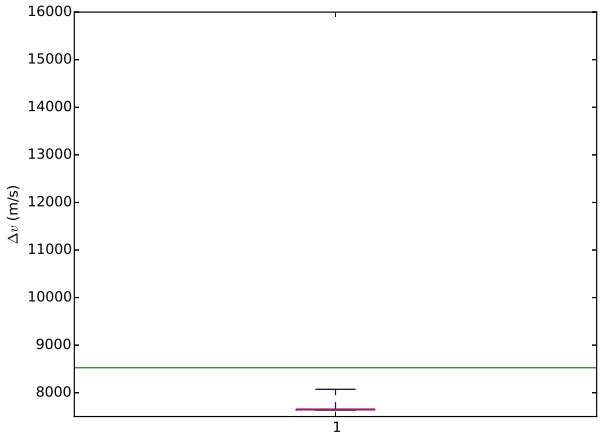
As we can see, the Cassini problem proves again to be the one with the most potential for this type of application, whose solutions are the most impaired by the time discretization. Even so, the final Δv values obtained by this method are much worse than the ones found via the Initial Guess one. Theoretically, this algorithm could be able to find better solutions than the combinatorial one; we know, for instance, that the best solution with limited path length (in Tables 4.1 and 4.2) is superior in quality to the combinatorial case (in Tables 3.1 and 3.2). Moreover, there is potential to find solutions of greater path length while still observing the constraint in mission duration, due to the optimized times. However, the application of the method does not live up to the suppositions; since the amount of time for a run is limited, the algorithm does not have time to expand the tree to greater sequence lengths. Furthermore, the best solutions found are the ones with the *popsize* value of 15, proving that increasing this number, while providing a more throughout search, makes each optimization pass slower.

We have decided to also evaluate this method in regards to its time performance. After profiling the algorithm, we found that the minimization part totaled 99.982% of the entire time duration. Of this, 90.358% was spent in computations inside the function to be minimized – the differential evolution routine took less than 10% of it. This is easily understood by the fact that the differential evolution method, as explained in Section 4.3.1, is a very simple one, much more so than the Limited-Memory Bounded BFGS – thus, it does not spend much computational cost. However, since there is no initial guess for this method, the time intervals for the times of flight have to be explored in a much more throughout manner, which demands more ephemeris and Δv computations and, finally, a bigger cost in terms of the function to minimize. Being so, this algorithm explores even less of the tree than the Initial Guess one and leaves

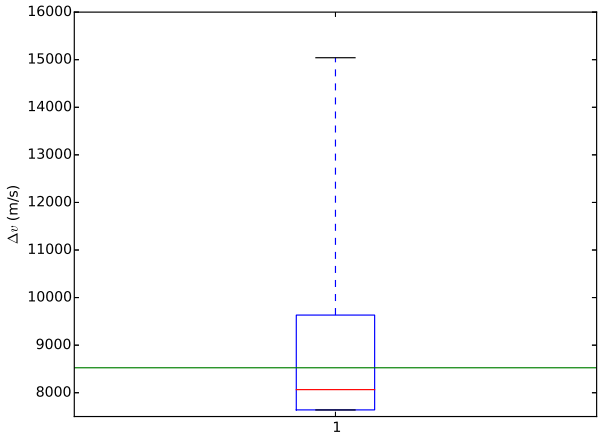


(a) Rosetta Problem with Population size of 15

(b) Rosetta Problem with Population size of 169



(c) Cassini Problem with Population size of 15



(d) Cassini Problem with Population size of 169

Figure 4.9: Final Δv Results for a budget of 1 Hour using an ϵ-Greedy policy for the Optimization with Differential Evolution

out many promising flyby sequences, although explores the few that obtains in a more extensive manner than that method.

In this way, we can infer that the Initial Guess method is better than the Global Minimization one; with the same constrains in budget, it can deliver better results, due to the fact that the optimizer takes less time to run - it has an initial guess of the result, and only searches in a limited space around it. The Global Minimization method, although possessing the potential to dig into solutions that might have been overlooked by the discretization, such as the best solution represented in Section 4.3.3, loses in terms of the optimization function.

For these test cases, it is then considered that the MCTS used with a time grid is the best way to get fast, reliable results. However, if not constrained in terms of budget, both hybrid optimization methods can deliver the results with the smallest fuel consumption. In order to further argue for this allegation, we have computed the final Δv solutions of both optimization methods that correspond to the best sequences of the discrete search tree, whose results are presented next.

Best Tree Sequences

On this Section, we have decided to implement the optimizer only on the solution given after the budget is depleted, instead of doing so at the end of every simulation. In this way, the implemented algorithm is the MCTS with the applied time grid but, almost at the end of the time budget, the optimization is performed on the resulting path.

The two optimization methods previously described are here compared (L-BFGS-B and Differential Evolution) and the results are depicted on Figures 4.10 and 4.11. These show the solution from ten runs of the combinatorial case (the green points) and the corresponding results in both optimization methods (red colored points for the Differential Evolution method, blue ones for the L-BFGS-B).

The total run time was, again, of one hour, for both the Rosetta and Cassini test cases. The Tree Policy used was the ϵGreedy, with the parameters declared in Section 3.3.3.

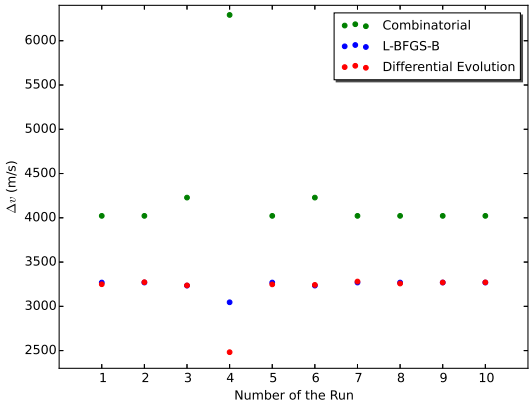


Figure 4.10: Optimization of the Final Δv of the Combinatorial Solution for the Rosetta Problem, using the L-BFGS-B and the Differential Evolution Methods

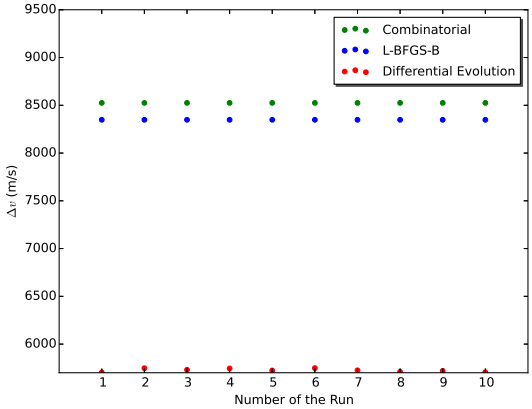


Figure 4.11: Optimization of the Final Δv of the Combinatorial Solution for the Cassini Problem, using the L-BFGS-B and the Differential Evolution Methods

As previously assumed, the solutions found by the hybrid optimization techniques are always better than the original combinatorial one. With these results, it is difficult to infer what the best method is.

The Differential Evolution algorithm, although taking longer (something we measured computationally), also searches the state space more thoroughly. This is evident on the Cassini case: the L-BFGS-B with the discretized times as an initial guess finds a solution close to the best combinatorial one, but the global optimization delivers much better Δv using points far from the considered hypothesis. This indicates us that, as previously concluded, the time discretization does not seem to benefit the Cassini case, since we can infer there are much better solutions in between the points of the time grid.

This is not, however, the case of Rosetta: the discretization seems to be fine enough so that the local optimization finds a better solution, but not by a significant amount of Δv. Both optimization methods perform similarly, but the solutions found by the L-BFGS-B yield always the same Δv, in opposition to the Differential Evolution method, which does not always provide the same solution due to the random sampling of the initial population.

Asteroid Problem

All the Tree Policies were tried in this case, to assert the possibility of changes in behavior from the discrete problem. The values of ϵ and C_P are the same ones used in the combinatorial problem (respectively, 0.0001 and 0.04), since the reward function and objectives are the same. The results are depicted in the following figures in the form of boxplots in Figure 4.12. As initially proposed, all of these solutions were confirmed, by our model, to be physically correct.

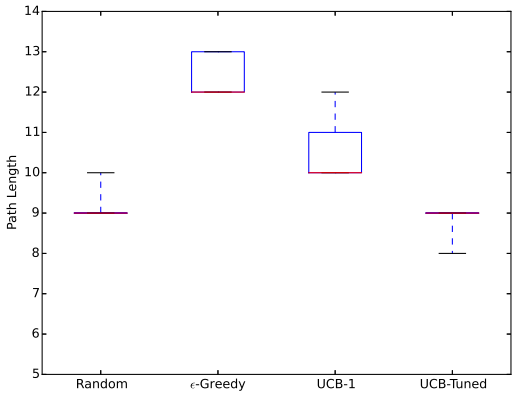


Figure 4.12: Different Tree Policies for the Hybrid Optimization of the Asteroid Problem

We can see that we have managed to get slightly better results than on the combinatorial problem, due to a better time management, since all of the sequences are constrained by time: not one more asteroid can be added or the mission duration would be exceeded. There are not many differences in terms of Policy Comparison, but the ϵ-Greedy Policy excels again on this case, managing sequences of up to 13 asteroids and maintaining the highest median out of all Tree policies. The others compare to the ϵ-Greedy one in almost the same way as in the combinatorial problem, in Section 3.3.4: the UCB-1 is the second best, the Random and the UCB-Tuned perform similarly. We can infer that the parameters remain adequate for this problem.

In conclusion, the hybrid problem, with an ϵ-Greedy policy, yields better solutions than the combina-

torial one, but still leaving space for improvement in regards to the optimization objective.

Chapter 5

Conclusions and Future Work

During this work, we have studied the implementation of Monte Carlo Tree Search algorithms in the domain of Interplanetary Trajectory design. These methods are very versatile in problems that have the form of tree searches, are aheuristic and show great potential in different fields of application. Interplanetary trajectory design can be interpreted as a single-player game that does not possess any reliable heuristic for a more efficient traversal of the tree, so the algorithm presented itself as an option worth studying. Furthermore, it is a method capable of yielding very good results in a short time span, something very interesting in space mission design.

The problem of finding the best trajectory between two bodies consists simply on developing the best combination of planets and times of flight in between them, with different purposes depending on the test cases. In the Rosetta and Cassini problems, the objective was to minimize the total amount of Δv required for the trip; in the Asteroid one, the goal was to maximize the number of asteroids visited in one sequence only.

There is a discrete set of planets that may be selected to this path; however, the times of flight belong to a continuous interval. In this way, two approaches were implemented: to discretize the times of flight depending on the distances between planets, or to leave this value as the design variable of a bounded optimization algorithm, constrained by bounds. These strategies were compared in order to check which yielded the best results depending on the goal.

In spite of not relying on any heuristic, for the algorithm to learn and to efficiently balance the exploration and exploitation of the tree, several parameters had to be tuned. For this problem, we determined the maximum Δv for a trajectory to be considered and also the parameters ϵ and C_P, necessary for the implementation of different Selection Policies.

Considering that the MCTS is an algorithm with the potential to find very good results in a short time period, not being an exhaustive search method, both of these strategies were compared with a time restriction of one hour, but also with a limit in the number of simulations the algorithm can perform, to check possibilities of lessening its computational effort.

Overall, we have concluded that the MCTS algorithm is sensitive to the tuning of its parameters, performing much poorly when the learning is not done in the best way possible. In the end, a finely

tuned ϵ-Greedy Policy performs better than all the other policies, while also being the fastest policy considered that involves learning, for both types of problems - even so, the parameters tuned lie in an interval inside which they can variate.

Considering the Rosetta and Cassini cases, the results obtained from both the discrete and the hybrid problem can be interpreted in two ways, depending on our purpose with the trajectory search. The time discretization made on the first case provided a finite search tree whose best result was a fixed node - the MCTS algorithm with the ϵ-Greedy Policy found this value most of the time, given the constrained time budget, and is very good performance-wise.

On the other hand, the hybrid problem, dealing with the optimization of a continuous interval, has the potential of finding better solutions on the state space. However, the amount of time spent by the optimizer on the entire run takes more than 98% of the designated budget. While using the Differential Evolution method, this is even more noticeable than when using the L-BFGS-B algorithm, since the search space of the former is bigger than the latter, which can rely on an initial guess for the minimum. Yet, without a reliable assumption for this value, the L-BFGS-B algorithm performs very poorly, getting easily trapped in local minima. The Differential Evolution method, while slower, provides for a better coverage of the state space and quality of minima, although needing a more throughout definition of function parameters that may make the algorithm even slower. The L-BFGS-B technique is preferable when the initial hypothesis comes from a well discretized problem, as it happens on the Rosetta case.

Ultimately, a compromise method is to utilize the optimizer on the best discrete solution, only at the end of the run. It may not be the best hybrid solution of the state space, but it will always be better than the one found by the combinatorial tree. This, nevertheless, cannot be done on the Asteroid case.

In the Asteroid problem, there can be no improvement on the top of a solution, due to the constraints on the search tree. The purpose is to find an adequate time discretization that allows us to include the greater number of asteroids possible in one path, while always complying to the mass constraints. We have determined that a heavier spacecraft is less maneuverable than a lighter one, and therefore it takes longer for a given trajectory to be performed in feasible terms. As such, the times of flight will start out as being bigger than in the end of the sequence, when the spacecraft has spent part of its fuel mass. In the combinatorial case, we have concluded a Logarithmic time distribution, with the greater concentration of points in the middle of out interval of feasibility, to be the best, eliminating too fast and too slow time values. Yet, the implementation of the Differential Evolution function in the scope of this problem proves serve this purpose better, using again the ϵ-Greedy policy.

The optimization of the spacecraft's mass in order to get the greatest possible sequence lengths in the Asteroid problem was the solution presented in this project; nevertheless, these two goals are not equivalent and, as such, we suggest that an specially adapted algorithm for the sequence maximization that includes the trajectory's length as the variable to optimize would yield better results. Also, the increase of the pool of bodies to visit would be a good way to test the Monte Carlo Tree Search's effectiveness and speed against problems with a very large scope.

We have deemed the Monte Carlo Tree Search algorithm to be an efficient alternative for trajectory design, yielding very good results in a very short time period, when using a finely tuned Tree Policy - the

tuning method is of utmost importance. A time discretization is of benefit to this problem, but it has to be adequate for its dimension. Moreover, engineering constraints need to be carefully thought of, since they change the problem dramatically. Furthermore, applying an optimizer at the end of the budget on the discrete problem is a better option than optimizing each solution. In conclusion, the most important consideration to be taken is that the real mission trajectories were found for both cases, besides the Δv-EGA maneuver on the Rosetta mission path which is not included in the model.

In the study of the Monte Carlo Tree Search, several variants are still left to explore relative to this domain. Single-player Monte Carlo Tree Search is fairly unexplored, and other learning mechanisms could be worth implementing, such as other Tree and Selection Policies. Also, the HOOT algorithm, while performing very poorly, could use an exhaustive tuning of its several parameters (δ and the budget of HOO). Furthermore, taking into account the importance of the time discretization, the development of different grids in the scope of planetary flight might be worth considering.

In regards to the model utilized to simulate the Space mechanics and maneuvers, we propose some extensions to be made, concerning the addition of low energy transfers, multiple revolution gravity assists and Δv leveraging maneuvers, such as the Δv-EGA included in the Rosetta mission path. These additions will make the model more accurate in depicting the possibilities presented by orbital mechanics in regards to each mission design.

Bibliography

[1] Sylvain Gelly and David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175, 2011.

[2] Jeroen Melman. Trajectory Optimization for a Mission to Neptune and Triton. Master's thesis, TUDelft, the Netherlands, 2007.

[3] Daniel Hennes and Dario Izzo. Interplanetary Trajectory Planning with Monte Carlo Tree Search. *IJCAI'15*, 2015.

[4] NASA Master Catalog, Venera 1 Mission. `http://nssdc.gsfc.nasa.gov/nmc/masterCatalog.do?sc=1961-003A`. Accessed: 2015-09-10.

[5] NASA Master Catalog, Voyager 1 Mission. `http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1977-084A`. Accessed: 2015-09-10.

[6] NASA Master Catalog, New Horizons Mission. `http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=2006-001A`. Accessed: 2015-09-10.

[7] Luke W. McNamara and Robert D. Braun. Definition of the Design Trajectory and Entry Flight Corridor for the NASA Orion Exploration Mission 1: Entry Trajectory using an Integrated Approach and Optimization. *American Astronautical Society Guidance and Control Conference*, 2014.

[8] Jacob Williams, Juan S. Senent, Cesar Ocampo, Ravi Mathur, and Elizabeth C. Davis. Overview and Software Architecture of the Copernicus Trajectory Design and Optimization System. *4th International Conference on Astrodynamics Tools and Techniques, Madrid, Spain*, 2010.

[9] Jacob Williams, Juan S. Senent, Cesar Ocampo, and David E. Lee. Recent Improvements to the Copernicus Trajectory Design and Optimization System. *Advances in the Astronautical Sciences*, 2012.

[10] Steven P. Hughes. General Mission Analysis Tool (GMAT). `http://gmatcentral.org/`, 2007.

[11] James M Longuski and Steve N Williams. Automated Design of Gravity-Assist Trajectories to Mars and the Outer Planets. *Celestial Mechanics and Dynamical Astronomy*, 52(3):207–220, 1991.

[12] Guillermo Ortega, Chris Laurel, Sven Erb, Carlos Lopez, Christof Bueskens, Erwin Mooij, Roland Klees, Fernando Lau, Paulo Gil, Joerg Fliege, et al. STA, the Space Trajectory Analysis Project. *4th International Conference on Astrodynamics Tools and Techniques*, 2010.

[13] Systems Tool Kit STK. http://www.agi.com/products/stk/. Accessed: 2015-09-20.

[14] Andrew F Heaton, Nathan J Strange, James M Longuski, and Eugene P Bonfiglio. Automated Design of the Europa Orbiter Tour. *Journal of Spacecraft and Rockets*, 39(1):17–22, 2002.

[15] Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. *Machine Learning, Springer*, 2006.

[16] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1):57–83, 2002.

[17] Stockfish: Strong open source chess engine. https://stockfishchess.org/. Accessed: 2015-09-10.

[18] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, 2009.

[19] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, and Yau-Hwang Kuo. A Novel Ontology for Computer Go Knowledge Management. *IEEE FUZZ*, 2009.

[20] Tapani Raiko and Jaakko Peltonen. Application of UCT Search to the Connection Games of Hex, Y, *Star, and Renkula! *AI and Machine Consciousness*, 2008.

[21] Spyridon Samothrakis, David Robles, and Simon M. Lucas. A UCT Agent for Tron: Initial Investigations. *Computational Intelligence and Games*, 2010.

[22] Tristan Cazenave. Nested Monte-Carlo Search. *IJCAI'09*, 2009.

[23] Christopher D. Rosin. Nested Rollout Policy Adaptation forMonte-Carlo Search. *IJCAI'11*, 2011.

[24] Dario Izzo. PyGMO and PyKEP: Open Source Tools for Massively Parallel Optimization in Astrodynamics. *Fifth International Conference on Astrodynamics Tools and Techniques*, 2002.

[25] Dario Izzo. Revisiting Lambert's Problem. *Celestial Mechanics and Dynamical Astronomy, Springer*, 2014.

[26] SPICE: An Observation Geometry System for Planetary Science Missions. http://naif.jpl.nasa.gov/naif/index.htm/. Accessed: 2015-10-03.

[27] Georg Hohmann. Der Hallux valgus und die übrigen Zehenverkrümmungen. *Ergebnisse der Chirurgie und Orthopädie, Springer Berlin Heidelberg*, 1925.

[28] Adel S. Solimanc Badaoui El Mabsout, Osman M.Kamel. The Optimization of the Orbital Hohmann Transfer. *Acta Astronautica*, 2009.

[29] R. A. Broucke. The Celestial Mechanics of Gravity Assist. *AIAA Journal*, 1988.

[30] Damon F. Landau and James M. Longuski. Trajectories for Human Missions to Mars, Part II: Low-Thrust Transfers. *Journal of Spacecraft and Rockets*, 43(5):1043–1048, 2006.

[31] Johan Schoenmaekers and Rainer Bauske. Re-design of the Rosetta Mission for Launch in 2004. In *18th International Symposium on Space Flight Dynamics*, volume 548, page 227, 2004.

[32] Chit Hong Yam, T. Troy McConaghy, K. Joseph Chen, and James M. Longuski. Design of Low-Thrust Gravity-Assist Trajectories to the Outer Planets. In *55th International Astronautical Congress*, 2004.

[33] TieDing Guo, FangHua Jiang, HeXi Baoyin, and Li JunFeng. Fuel Optimal Low Thrust Rendezvous with Outer Planets via Gravity Assist. *Science China Physics, Mechanics and Astronomy*, 54(4):756–769, 2011.

[34] GTOC 7 – Multi-Spacecraft Exploration of the Asteroid Belt. `http://sophia.estec.esa.int/gtoc_portal/?page_id=515`. Accessed: 2015-10-03.

[35] Walter D Pullen. Think Labyrinth: Maze Algorithms. *See http://www. astrolog. org/labyrnth/algrithm. htm*, 1989.

[36] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[37] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite Time Analysis of the Multiarmed Bandit Problem. *Machine Learning, Springer*, 2002.

[38] Ashish Sabharwal, Horst Samulowitz, and Chandra Redd. Guiding Combinatorial Optimization with UCT. *Springer*, 2012.

[39] Tristan Cazenave. Reflexive Monte-Carlo Search. *Proceedings of Computer Games Workshop*, 2007.

[40] Matthew L. Ginsberg. GIB: Imperfect Information in a Computationally Challenging Game. *Journal of Artificial Intelligence Research*, 2011.

[41] Peter Auer. Using Confidence Bounds for Exploitation-Exploration Trade-offs. *The Journal of Machine Learning Research*, 2003.

[42] Ari Weinstein, Chris Mansley, and Michael Littman. Sample-based Planning for Continuous Action Markov Decision Processes. *ICAPS*, 2011.

[43] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-Armed Bandits. *Journal of Machine Learning, Springer*, 2011.

[44] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, LTD, first edition, 2001.

[45] M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 1979.

[46] Rainer Storn and Kenneth Price. Differential evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 1996.

[47] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 1995.

[48] Dieter Kraft. A Software Package for Sequential Quadratic Programming. Technical Report DFVLR-FB 88-28, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, 1988.

[49] Stephen G. Nash. Newton-type Minimization via the Lanczos Method. *SIAM Journal on Numerical Analysis*, 21(4):770–788, 1984.

[50] Michael Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

[51] Matthew Dixon and Mohammad Zubair. Calibration of Stochastic Volatility Models on a Multi-Core CPU Cluster. *WHPCF Proceedings of the 6th Workshop on High Performance Computational Finance*, 2013.

[52] The Minor Planet Center. `http://www.minorplanetcenter.net/`. Accessed: 2015-10-13.

Appendix A

Asteroids Data

Asteroid Number	Name	Asteroid Number	Name	Asteroid Number	Name
523	Aletta	658	Mimosa	752	Luxembourg
754	Haworth	786	Swissair	932	Lujiaxi
973	Onnie	1151	Gibson	1554	Kotelnikov
1892	Arpetito	1913	Hagihara	1917	WV9
2640	Nobuhisa	2658	UD8	2788	Aarhus
2814	Timiryazev	3178	Jo-Ann	3574	Huangsushu
3836	CJ	3855	UW21	3874	Marioferrero
3932	AN233	4066	Risehvezd	4439	VL3
5032	WE12	5402	AO23	5758	ET45
5765	RJ10	7713	Creighton	7879	Hieizan
7889	QC12	8255	PU	8426	Kupe
8487	YT8	8499	LF2	8511	GB36
8868	Edwardsu	8978	Sigmund	9224	Takarajima
9622	Sauer	9634	QU47	10533	Laguerre
10751	Goven	11069	RE2	11088	T-3
11557	Murom	11954	HY77	12475	QD38
12489	QZ43	12508	Benner	12526	TN33
12548	Anmaschlegel	12572	DW	13836	Masiero
13959	T-3	14167	QB7	14248	XQ
14983	Carlyrosser	15039	AT2	15381	AB10
15681	Elainemccall	15773	AD14	15776	HX45
15948	QX35	16000	Karlin	16005	EO26
16040	Fionapaine				

Table A.1: Asteroid Names from the Minor Planet Center [52] and their respective numerical Representation in the Algorithm