UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

# Genetic algorithm for multi-gravity assist interplanetary trajectory optimisation

**TFM TITLE: Genetic algorithm for multi-gravity assist interplanetary trajectory optimisation**

**DEGREE: Master of Aerospace Science and Technology**

**AUTHOR: Iker Diaz Cilleruelo**

**ADVISOR: Sebastian Andreas Altmeyer**

**DATE: February 6, 2023**

**Title :** Genetic algorithm for multi-gravity assist interplanetary trajectory optimisation.
**Author:** Iker Diaz Cilleruelo

**Advisor:** Sebastian Andreas Altmeyer

**Date:** February 6, 2023

**Overview**

This master thesis presents the development of a genetic algorithm for optimizing multi-planetary gravity assist trajectories. The algorithm was designed to address the challenges of finding the most efficient trajectory for a spacecraft traveling into deep space using multiple planets while performing gravity-assist maneuvers. Given a model for the interplanetary trajectory, the algorithm is able to find a feasible optimal solution. The proposed approach was tested on a set of real missions and was shown to produce solutions more optimal than the real ones given our trajectory model. The results demonstrate the effectiveness of evolutionary algorithms, in concrete genetic algorithms, in finding optimal multi-planetary gravity assist trajectories, making it a valuable tool for mission planning in space exploration.

**Título:** Algoritmo genético para la optimización de trayectorias interplanetarias con multiple assistencia gravitatoria
**Autor:** Iker Diaz Cilleruelo

**Director:** Sebastian Andreas Altmeyer

**Fecha:** 6 de febrero de 2023

**Resumen**

Esta tesis de máster presenta el desarrollo de un algoritmo genético para optimizar trayectorias multiplanetarias con assitencia gravitatoria. El algoritmo ha sido diseñado con el objetivo de encontrar la trayectoria más eficiente para missions entre múltiples planetas en las cuales se realizan maniobras de asistencia gravitatoria. Dado un modelo para calcular la trayectoria interplanetaria, el algoritmo es capaz de encontrar una solución óptima y factible. El enfoque propuesto se ha probado en un conjunto de misiones reales y se he demostrado que produce soluciones más óptimas que las reales dado nuestro modelo de la trayectoria. Los resultados demuestran la eficacia de los algoritmos evolutivos, en concreto de los algoritmos genéticos, para encontrar trayectorias óptimas de asistencia gravitatoria multiplanetaria, lo que los convierte en una valiosa herramienta para la planificación de misiones de exploración espacial.

# CONTENTS


**CHAPTER 1. Introduction** . . . . . 1

**CHAPTER 2. Interplanetary trajectory** . . . . . 5

**2.1. Transfer orbits** . . . . . 5
2.1.1. Hohmann transfer . . . . . 5
2.1.2. Bi-elliptic Hohmann transfer . . . . . 6

**2.2. Patched conic** . . . . . 6

**2.3. Planetary gravity assist flyby** . . . . . 8
2.3.1. Departure . . . . . 8
2.3.2. Flyby maneuver . . . . . 9

**2.4. Historical uses of MGA** . . . . . 10
2.4.1. Voyager I and II . . . . . 11
2.4.2. Galileo . . . . . 12
2.4.3. BepiColombo . . . . . 13

**CHAPTER 3. Evolutionary algorithms** . . . . . 15

**3.1. Genetic algorithms** . . . . . 15
3.1.1. Genetic operators . . . . . 17

**CHAPTER 4. Algorithm formulation** . . . . . 21
4.0.1. Overview . . . . . 21

**4.1. Trajectory design** . . . . . 22
4.1.1. Ephemeris . . . . . 22
4.1.2. Lambert solver . . . . . 25
4.1.3. Patched conic . . . . . 26

**4.2. Genetic algorithm** . . . . . 28
4.2.1. Objective function and constraints . . . . . 28
4.2.2. Genetic evolution structure . . . . . 30
4.2.3. Multithreading . . . . . 32

**CHAPTER 5. Test cases and results** . . . . . 35

**5.1. Real missions** . . . . . 35

5.1.1. Voyager-I mission . . . . . 35
5.1.2. Voyager-II mission . . . . . 38
5.1.3. Galileo mission . . . . . 41

**5.2. Genetic algorithm parameters . . . . . 44**
5.2.1. Population and generations . . . . . 44
5.2.2. Genetic operators . . . . . 45

**CHAPTER 6. Conclusion . . . . . 47**

**6.1. Future work . . . . . 48**

**Bibliography . . . . . 51**

# LIST OF FIGURES


2.1 Hohmann transfer orbit (from [1] Chap. 6, Figure 6.2) . . . . . 5
2.2 Bi-elliptic Hohmann transfer orbit (from [1] Chap. 6, Figure 6.9) . . . . . 7
2.3 Planetary departure under the sphere of influence of a planet (from [1] Chap. 8, Figure 8.8) . . . . . 8
2.4 Trailing-side flyby (from [1] Chap. 8, Figure 8.19) . . . . . 9
2.5 Voyager I and II interplanetary trajectories (from [2] Figure 1-4) . . . . . 11
2.6 Voyager II velocity evolution (from [2] Figure 11-6) . . . . . 12
2.7 Galileo mission interplanetary trajectory (from [3]) . . . . . 12
2.8 BepiColombo mission interplanetary trajectory (from [4]) . . . . . 13

3.1 Illustration of roulette selection. . . . . 17

4.1 Code diagram . . . . . 21
4.2 Planetary orbit in heliocentric frame reference (from [1] Fig. 8.25) . . . . . 24
4.3 Diagram of the genetic algorithm . . . . . 30

5.1 Optimal trajectory vs trajectory of Voyager-I. . . . . 37
5.2 Voyager-I optimal trajectory speed evolution. . . . . 37
5.3 Fittest individual evolution for Voyager-I mission. . . . . 38
5.4 Optimal trajectory vs trajectory of Voyager-II. . . . . 40
5.5 Voyager-II optimal trajectory speed evolution. . . . . 41
5.6 Optimal trajectory vs trajectory of Galileo. . . . . 43
5.7 Galileo optimal trajectory speed evolution. . . . . 43
5.8 Compassion of 10 run solutions: 25 generations with 1500 individuals in the left and 15000 in the right. . . . . 44
5.9 Convergence of Voyager-I's mission for different population sizes. . . . . 45
5.10 Comparison of selection methods for Voyager-I. . . . . 46

# LIST OF TABLES

2.1 Timeline of Voyager I events. . . . . 11
2.2 Timeline of Voyager II events. . . . . 12
2.3 Timeline of Galileo events. . . . . 13

3.1 Tournament selection method illustration. . . . . 18

4.1 Keplerian elements and rates (valid from 1800 - 2500). Table from [5] . . . . . 23

5.1 Trajectory optimization for Voyager-I; inputs and solution. . . . . 36
5.2 Trajectory optimization for Voyager-II; inputs and solution. . . . . 39
5.3 Trajectory optimization for Galileo; inputs and solution. . . . . 42

# CHAPTER 1. INTRODUCTION

While the golden age of space exploration with pioneering missions such as the Mariners[6], the Voyagers[7], Galileo[8], and many others has ended, space exploration is certainly not something from the past. Space exploration is still nowadays at the forefront of the space and aerospace industry, with many missions such as BepiColombo[9], New Horizons[10], Juno[11], etc. While these kinds of missions are extremely complex and come up with many new problems, one of the most common and notable among all of them, is how to get the spacecraft to its destination. While it is something that has been solved from the early days, it is unique for every mission, and there are several ways one can tackle and solve the problem. However, the most common way of solving it, i.e. reaching another body of the solar system, is through the use of gravity assist maneuvers. This maneuvers are used to optimize the delta-v required for the trajectory, which results in less mass and fuel consumption. Some missions, will not even be possible without these maneuvers.

Gravity assist or flyby is a maneuver through which the traveling spacecraft does a close approach to the planet or body within the travel trajectory to increase the spacecraft's speed. In other words, this technique can add or subtract (depending on the relative direction) momentum to the spacecraft in question (relative to the sun). A more in-depth analysis of gravity assist maneuvers will be performed in the second section.
Usually, spacecraft use this technique to propel themselves to faraway bodies, where the required delta-v to reach them is too high. However, it can also be used to slow down a spacecraft before a rendezvous or an injection orbit in the target body or a close one. This technique was even required for the Voyager-I mission to reach its destination (Saturn), due to the launch vehicle (Titan III-E/Centaur D-IT [12]) only being able to provide enough delta-v to reach Jupiter. To exemplify how powerful this technique is, Voyager-I has nowadays reached "interstellar space", hence escaping the solar system [13].

While the process of a gravity assist maneuver is relatively well understood, the complexity from a mission design perspective is how to find the most ideal trajectory to reach the desired destination, using multiple gravity assists (MGA) maneuvers. Finding a viable interplanetary trajectory, while not trivial, can be done with a bit of effort. The problem comes when trying to find the most optimum trajectory; understanding as the most optimum one, the one with less overall required delta-v. Other factors can be considered when determining the optimally of a trajectory, for instance, the travel time. However, the most determining factor in the design of a mission's trajectory is the required delta-v. The cost of launching a mission is notably tight to the amount of delta-v required to be provided to the spacecraft. Usually, the higher the delta-v required, the bigger the launcher needed, and thus the cost. In addition, the most delta-v required during the flight phase, the more mass and complex the spacecraft will be, indirectly affecting the other spacecraft's subsystems. Furthermore, the amount of delta-v that can be provided to the spacecraft is limited by the launcher itself, e.g. the Titan/Centaur for Voyager-I. For this reason, trajectory design optimizers, mainly drive down the delta-v required, and other factor are set as secondary objectives.

Optimizing a trajectory can be defined as an NP-Hard problem. While the search space is extremely big; available bodies for flybys and time of flybys, some constraints are added to

reduce the complexity of the optimization problem. In many cases, the problem is divided into two sub-problems. The first part determines the best sequence of planets in which to do a flyby, while the other part optimizes the dates of each flyby. Both optimizations are different, but not independent. The first one is used as the setup for the second. After the sequences of planets have been decided, the second part of the algorithm optimizes the trajectory for the given sequence. This process is repeated until there is a convergence for the best sequence and trajectory [14][15]. This kind of approach complicates the problem. To reduce the complexity, some simplifications are made in the form of constraints. Bounds are added to the problem to reduce the search space, for instance, the number of planets in which to do a flyby, as well as the possible arrangement of those. Other constraints in the form of bounds are added to the duration of the legs in the trajectory. These kinds of constraints are dangerous as they can prune out the optimal trajectory. For this reason, the pruning phase has to be done carefully, and constraints have to be large and permissive enough to reduce to possibility of pruning the optimal solution.

Together with the MGA trajectory model, there is a sub-variant denoted MGA-DSM (Deep Space Maneuvers) that contemplates the use of deep space delta-v maneuvers to increase the spacecraft speed, whereas the MGA only allows a delta-v impulse at the perigee of the flyby trajectory. This sub-variant is more open, but more complex. Many of the optimizers that allow these maneuvers are limited to one DSM impulse per leg. Using this impulse can improve the trajectory, as it can allow to better setup the spacecraft for a flyby (gain more speed) or reduce the overall duration of the trip. However, these maneuvers can require the use of more delta-v during flight. The Cassini mission [16] used this kind of approach to reach Saturn. It performed 2 deep space maneuvers, in the legs Earth-Venus and Jupiter-Saturn.
In this thesis, we will only consider the MGA model, therefore, delta-v impulses will only be allowed at perigee passage during the flybys.

From now on forward, we will consider only the optimization regarding the second problem, i.e. finding the trajectory from the leg intervals. Therefore, the sequence of planets in the trajectory will be known beforehand, and will be added as an input to the problem. Because the sequence determination is not related to the time intervals, another algorithm could be added in the future to work in conjunction with the present one.
As we have mentioned, this is a global optimization problem, with variables and constraints. Such a problem can be defined as follows, where the variables to be optimized are $t_i$

$$\vec{X} = [t_1, t_2, \ldots, t_n]. \tag{1.1}$$

$t_i$ are the time events of the trajectory. The $\vec{X}$ vector contains all the mission's time intervals, where $t_1$ would be the departure date. Every $t_i, \forall i > 1$ is the date on which a flyby is performed. The last $t_n$ would be the arrival date. With this vector, the trajectory can be inferred by computing the different transfers between the planets at the corresponding times.
The objective function can then be expressed as

$$C = f(\vec{X}) + g(\vec{X}), \tag{1.2}$$

where $g(\vec{X})$ is a penalty function. The only hard constraints are in the form of bounds to the variables optimized

$$t_{min} < t_i < t_{max}. \tag{1.3}$$

Soft constraints are added to the penalty function.

This optimization problem can be solved by utilizing heuristic global optimization algorithms. Those algorithms are combined with deterministic search space pruning algorithms to reduce the complexity and improve performance [17]. However, these kinds of algorithms are not simple and required complex optimization techniques to solve them.
While these kinds of algorithms were the only ones feasible to be used in the back in the day, in the present, thanks to the advance in computational power, new kinds of algorithms can be used to optimize the trajectory. Evolutionary algorithms are such a kind. The approach used by these types of algorithms has been shown to work since the late 90s [18].
Evolutionary algorithms are a kind of optimization algorithms that work by modeling the principle of natural selection and evolution. While there are many variants, e.g. genetic algorithms, differential algorithms, particle swarm optimization, Gaussian adaptation, etc, they all rely on the same principles. They essentially simulate the natural process of evolution and selection of the fittest individuals to converge a certain population to the optimal solution.
This sort of approach requires a greater computational power when compared to deterministic optimization algorithms, which on the other hand are more complex to model and define. Deterministic optimization algorithms require more variables, constraints, and accurate modeling of the problems (functions, gradients, etc) which are hard to determine for nonlinear problems. On the other hand, evolutionary algorithms are much simpler, as they only need “solutions” to the problem.

In this thesis, we develop from the ground up a complete trajectory optimizer using a genetic optimization algorithm. The algorithm is developed in C++, with some parts in Python for visualization and other tools. We will analyze all the steps required to model an MGA interplanetary trajectory, from the computation of the ephemeris for the planets (position and velocity at a certain instant in time) to the computation of the transfer orbits (Lambert transfer) and the computation of the flybys. These values are then used in conjunction with a genetic algorithm. As well, we study and discuss the implementation of all the parameters and operations associated with a genetic algorithm, and how those affect the convergence of it. Later, we showcase how genetic algorithms can be used to find near-optimal MGA interplanetary trajectories at the expense of a bigger computational power, traded for a less complex algorithm when compared to deterministic optimizers. We complete the analysis by testing the algorithm with real cases missions such as the Voyagers and the Galileo.

# CHAPTER 2. INTERPLANETARY TRAJECTORY

We consider an interplanetary trajectory as the path that the spacecraft follows as it travels between different planets or other celestial bodies. As we have mentioned earlier, this trajectory must be designed properly to optimize the delta-v required by the spacecraft.

## 2.1. Transfer orbits

We consider transfer orbit as the trajectory between two given celestial bodies. The orbit can be arbitrarily defined, although there exist some standard ones such as Hohmann transfers, Bi-elliptic Hohmann transfers, and low-energy transfer orbits. Each one has its own characteristics that make them ideal in different scenarios.

### 2.1.1. Hohmann transfer

This kind of transfer is generally the most efficient one. It is the type transfer that requires less energy (delta-v) in the majority of cases [19]. It is a two-impulse maneuver for transfers between orbits that share the same focus point, i.e. orbit the same body. The transfer is conducted between the lower and higher points of the two orbits. The transfer is initiated with a burn at the perigee of the departure orbit (lower energy point) and it is finalized with a burn at the apogee of the target orbit. In figure 2.1 can observe the departure point as the perigee of the initial orbit in the inner circle. The transfer orbit ends at the apogee of the target orbit (the outer circle). The two burns are required to boost the spacecraft into the transfer orbit, which has more energy due to the increased semi-major axis. The same is needed to slow down to a lower energy target orbit. Otherwise, the spacecraft would come back to the departure orbit [1] Sect 6.2.

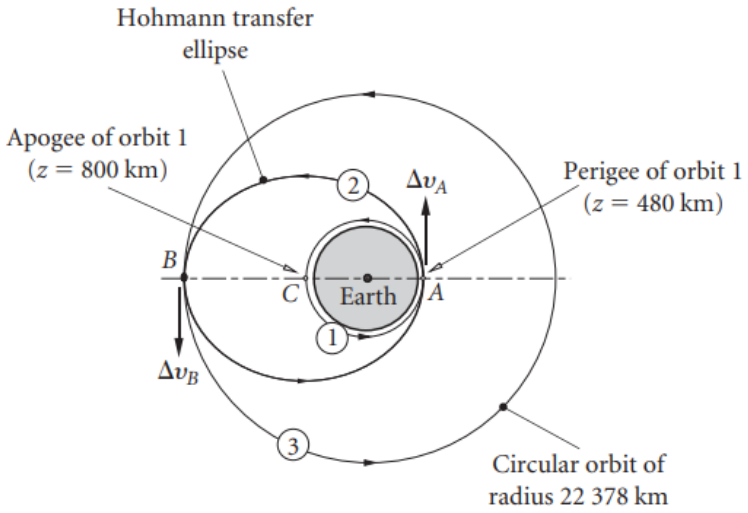


Figure 2.1: Hohmann transfer orbit (from [1] Chap. 6, Figure 6.2)

The delta-v required for a Hohmann transfer departing from a point $D$, in a planet with

circular orbital speed $V_1$ is given by [1] Eq. (8.3).

$$\Delta V_D = V_D^{(v)} - V_1 = \sqrt{\frac{\mu_{SUN}}{r_1}}\left(\sqrt{\frac{2r_2}{r_1+r_2}} - 1\right). \tag{2.1}$$

And the delta-v required at arrival is given by [1] Eq. (8.4).

$$\Delta V_A = V_2 - V_A^{(v)} = \sqrt{\frac{\mu_{SUN}}{r_2}}\left(1 - \sqrt{\frac{2r_1}{r_1+r_2}}\right), \tag{2.2}$$

where $\mu$ is the standard gravitational parameter of a celestial body, in this case $\mu_{SUN}$ is the one from the Sun ($1.327 \times 10^{20}$ [$m^3 s^{-2}$]).

### 2.1.2. Bi-elliptic Hohmann transfer

The Bi-elliptic Hohmann transfer uses the same principle as the Hohmann although with a small change. The pure Hohmann transfer has an ellipse form between two circular orbits. The Bi-elliptic orbit, on the other hand, is formed by two coaxial semi-ellipses that extend beyond the outer target orbit 2.2. In some cases, this approach can be more efficient than a simple Hohmann transfer. The Hohmann and Bi-elliptic Hohmann delta-v compare as follows [1] Eq. (6.4a) and (6.4b)

$$\Delta v_{Hohmann} = \left[\frac{1}{\sqrt{\alpha}} - \frac{\sqrt{2}(1-\alpha))}{\sqrt{\alpha(1+\alpha)}} - 1\right]\sqrt{\frac{\mu}{r_A}}$$

$$\Delta v_{Bi-elliptic} = \left[\sqrt{\frac{2(\alpha+\beta)}{\alpha\beta}} - \frac{1+\sqrt{\alpha}}{\sqrt{\alpha}} - \sqrt{\frac{2}{\beta(1+\beta)}}(1-\beta)\right]\sqrt{\frac{\mu}{r_A}} \tag{2.3}$$

$$\tag{2.4}$$

where,

$$\alpha = \frac{r_C}{r_A}, \qquad \beta = \frac{r_B}{r_A}. \tag{2.5}$$

Using both equations, it can be shown that the Bi-elliptic Hohmann transfer orbit is more efficient than the standard Hohmann transfer when the radius of the outer circular orbit $r_C$ is at least 15 times bigger than the inner radius $r_A$. For a radius smaller than 11.9 times, the standard Hohmann transfer is effectively more efficient. While between these two radii, it depends on the apogee radius [1] Sect. 6.4.

## 2.2. Patched conic

When dealing with interplanetary trajectories, it is important to understand the behavior and forces acting upon the spacecraft traveling between different celestial bodies. The patched conic approximation is a method used to simplify these trajectories, and the interaction between the spacecraft and the N-bodies around it [1] Sect 8.5.

The patched conic method assumes that when a spacecraft is outside the sphere of influence $r_{SOI}$ of a planet [1] Sect. 8.4, the spacecraft is only affected by the Sun's gravity,

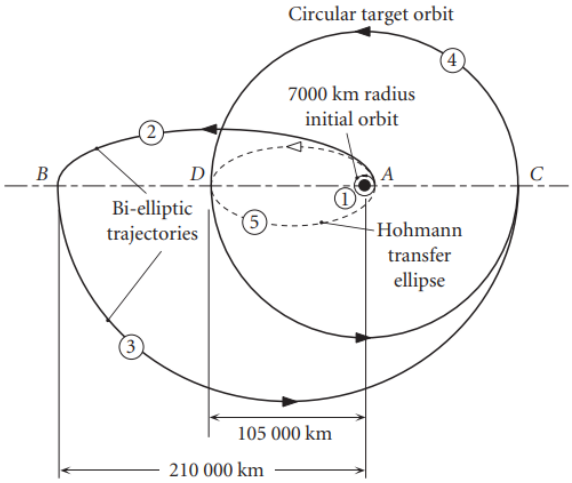


Figure 2.2: Bi-elliptic Hohmann transfer orbit (from [1] Chap. 6, Figure 6.9)

i.e. it follows an unperturbed Keplerian orbit around the Sun. Whereas when the spacecraft enters the sphere of influence, the spacecraft follows an unperturbed Keplerian orbit around the planet.
The sphere of influence is a region around a celestial body, usually, a planet, in which the gravitational force from the body on any object inside it is higher than the gravitational force of the Sun (bold circle in 2.3). For instance, on the surface of the Earth, the highest gravitational force, is the one exercised by the Earth, while in interplanetary space, the highest gravitational force is the one exercised by the Sun. The sphere of influence of a planet is defined as

$$r_{SOI} = R\left(\frac{m_p}{m_s}\right)^{\frac{2}{5}}. \tag{2.6}$$

This method can be used to simplify an N-body problem to a 2-body problem. While this kind of simplification is accurate enough in most cases (i.e. interplanetary trajectories), in reality, trajectories are affected by more than one body, and for cases where two bodies are relatively close to one another, this simplification cannot be made. As an example, in the Sun-Earth-Moon system, the 3-body problem must be considered, as so, Earth-Moon trajectories are much more complex [20].

In our case, the patched conic approximation is accurate enough as we will consider only interplanetary transfers (no interaction with other bodies as satellites), and the spacecraft will spend most of the time in a heliocentric orbit. The trajectory is broken down into 3 steps from a mission design perspective. If we consider the transfer between two planets, the heliocentric speeds are defined relative to the planets inside their sphere of influence (velocities at infinity), and relative to the Sun outside the spheres. The three relative speeds (departure, transfer, and arrival) are then determined and patched together.

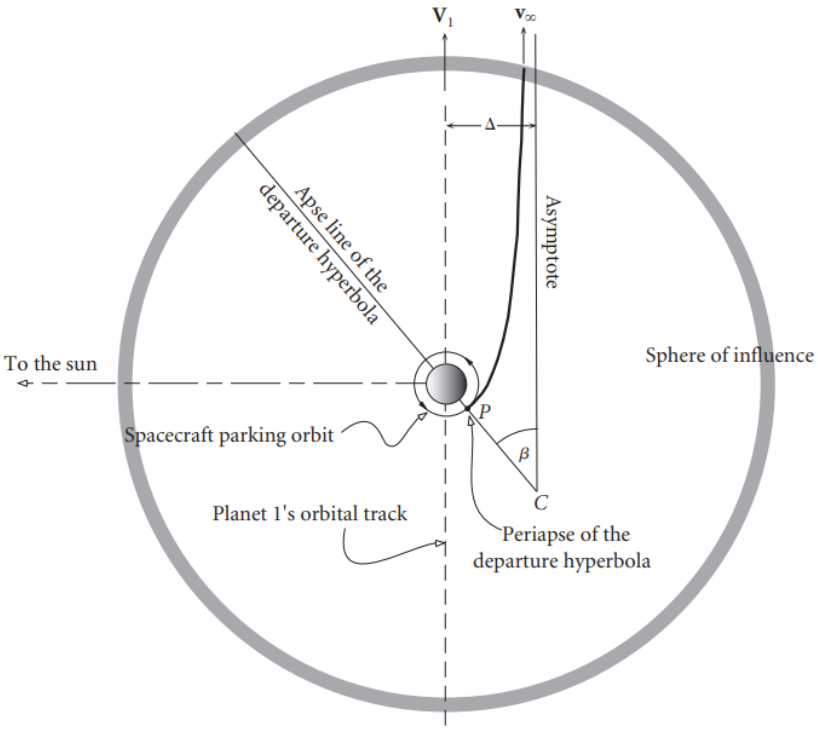


Figure 2.3: Planetary departure under the sphere of influence of a planet (from [1] Chap. 8, Figure 8.8)

## 2.3. Planetary gravity assist flyby

### 2.3.1. Departure

We consider an interplanetary trajectory that starts on a departure planet. In order for the spacecraft to begin its journey, it must escape the gravitational pull of the planet, i.e. have a hyperbolic speed relative to the planet higher than 0. In other words, the departure speed must be higher than the escape velocity of the planet.
To place the spacecraft in the desired transfer orbit, its heliocentric velocity must be parallel to the asymptote of the departure hyperbola at the sphere of influence crossing 2.3. The velocity at that point should match the desired hyperbolic velocity outside the sphere of influence.
In 2.3 we consider a departure from a parking orbit into a Hohmann transfer to an outer planet, i.e. further from the sun than the departure planet. In that case, the total amount of required delta-v to put the spacecraft into a hyperbolic departure trajectory is ([1] Eq. (8.42))

$$\Delta v = v_p - v_c = v_c \left( \sqrt{2 + \left( \frac{v_\infty}{v_c} \right)^2} - 1 \right), \tag{2.7}$$

where $v_p$ and $v_c$ are the speed at perigee and the circular speed respectively ([1] Eq. (8.40) and (8.42))

$$v_p = \frac{h}{r_p} = \sqrt{v_\infty^2 + \frac{2\mu_1}{r_p}}, \qquad v_c = \sqrt{\frac{\mu_1}{r_p}}. \tag{2.8}$$

### 2.3.2. Flyby maneuver

Once the spacecraft is in interplanetary space, the next steps are the flybys around other celestial bodies, normally, planets. During this maneuver, the spacecraft enters the sphere of influence of a planet and it exists with a new hyperbolic velocity. This change in velocity comes together with a deflection. The deflection is defined by a turning and $\delta$ at the perigee $P$ of the hyperbolic orbit. This change happens under the assumption that the perigee of the orbit is not smaller than the planet's radius, i.e. it does not impact the planet. In a similar manner, if the speed of the spacecraft is not sufficiently high, it may be captured by the planet's gravity. This case is usually desirable at the end of the trajectory when the spacecraft is aimed at orbiting the target planet.

From a planet's perspective, a spacecraft with a sufficiently large velocity (higher than the planet's $v_{esc}$) and perigee radius $r_p$, enters and exits the sphere of influence with the same velocity. In other words, there is no velocity gain. From a planet's perspective, considering that the spacecraft enters from the trailing-side, the speed of the craft will increase due to the planet's attraction and will decrease at the same rate after the perigee passage (at the planet's leading-side), essentially with a zero gain in velocity (ignoring possible atmospheric losses). The only change is a deflection on the hyperbolic trajectory 2.4. However, this situation from the Sun's perspective is different. When the spacecraft exists the sphere of influence, there is an excess of speed provided by the own planet's kinetic motion. Therefore, the heliocentric speed of the spacecraft is changed. Essentially, the spacecraft borrows speed from the body in which the flyby is performed. [1].

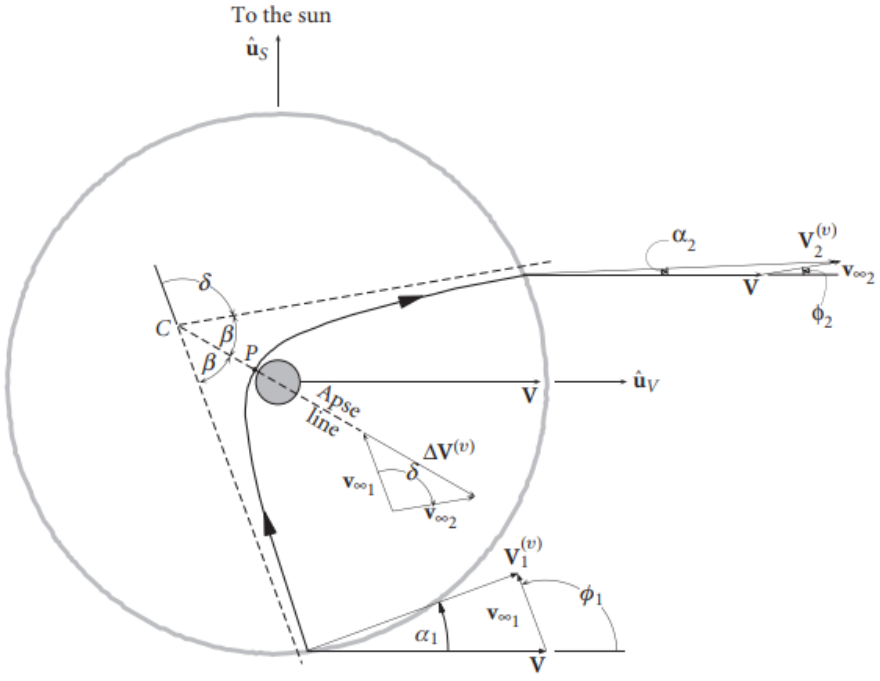


Figure 2.4: Trailing-side flyby (from [1] Chap. 8, Figure 8.19)

Using 2.4 where $\vec{V}_1$ and $\vec{V}_2$ are the heliocentric velocities at the inbound and outbound crossing points respectively, and $\vec{v}_\infty$ the heliocentric excess speeds, the gain in delta-v is

expressed as

$$\Delta\vec{V}^{(v)} = \vec{V}_2^{(v)} - \vec{V}_1^{(v)} = (\vec{V} + \vec{v}_{\infty_2}) - (\vec{V} + \vec{v}_{\infty_1}) \tag{2.9}$$

$$\Delta\vec{V}^{(v)} = \vec{v}_{\infty_2} - \vec{v}_{\infty_1} = \Delta\vec{v} \tag{2.10}$$

Mind that the delta-v can be positive or negative. If the flyby maneuver is done by entering through the leading-side of the planet's rotation, the speed will be decreased ((2.10) will be negative), as the spacecraft travels in the opposite direction. While if both go in the same direction, i.e. the spacecraft enters the trailing-side, the delta-v (2.10) will be positive. Consequently, the maneuver can be used to increase or decrease the spacecraft's velocity. In the solar system, planets orbit the Sun in a counterclockwise motion. Hence, the spacecraft's orbit should be in the same direction to gain speed. This is the most common case.

Mathematically, the flyby maneuver can be described with the following equations [1] Sect. 8.9. Let us consider an unpowered flyby. The relative hyperbolic excess velocities are computed as

$$\vec{v}_{\infty_1} = \vec{V}_1^{(v)} - \vec{V}_p \tag{2.11}$$

$$\vec{v}_{\infty_2} = \vec{V}_2^{(v)} - \vec{V}_p \tag{2.12}$$

where $\vec{V}_p$ is the planet's velocity and $\vec{V}_1^{(v)}$ and $\vec{V}_2^{(v)}$ the incoming and outgoing heliocentric velocities of the spacecraft. Using the scalar component form for $\vec{v}_{\infty_1}$, its magnitude is defined as ([1] Eq. 8.82)

$$v_\infty = \sqrt{\vec{v}_{\infty_1} \cdot \vec{v}_{\infty_1}} = \sqrt{\left[V_1^{(v)}\right]^2 + V^2 - 2V_1^{(1)} \cos\alpha_1}. \tag{2.13}$$

Then, we compute the eccentricity of the flyby parabola and the angular momentum as ([1] Eq. 8.83)

$$e = 1 + \frac{r_p v_\infty^2}{\mu} \qquad h = r_p \sqrt{v_\infty^2 + \frac{2\mu}{r_p}}. \tag{2.14}$$

Finally, using the eccentricity from (2.14), the turning angle can be computed as

$$\delta = 2\sin^{-1}\left(\frac{1}{e}\right). \tag{2.15}$$

It must be noted that in this case, we are considering an unpowered flyby. In our case, the flyby will be powered, with a delta-v maneuver allowed at the perigee. This maneuver is done to match the incoming velocity to the outgoing velocity. In section 4.1.3. we describe how these velocities are matched and the delta-v required computed. Essentially, we solve the delta-v requirements, turning angle, and perigee radius to patch the desired outgoing velocity to the incoming velocity.

## 2.4. Historical uses of MGA

Numerous exploration missions have used gravity-assist maneuvers to reach their destination. For missions adventuring beyond Mars or Venus, there is a notable potential on

using flybys to increase their velocity, reduce the delta-v required and potentially reduce the travel time. Almost every mission traveling farther than those planets have employed gravity-assist maneuvers. This section will briefly present some of the most notable missions.

### 2.4.1. Voyager I and II

The first mission that properly performed a gravity assist maneuver was the Mariner 10, however, the two Voyagers missions followed it soon after. Voyager I and II were aimed at studying Saturn and Jupiter, and Uranus and Neptune respectively. Voyager II was the first one to launch on the 20th of august 1977 on board a Titan III-E/Centaur D-IT. Voyager I was launched two weeks after on the 5th of September 1977, on board the same rocket. Both missions followed the same interplanetary trajectory until Saturn; they performed a flyby at Jupiter and Saturn. Voyager II continued further with a flyby at Uranus and Neptune 2.5. While Voyager I went into an interstellar trajectory after its encounter with Saturn.

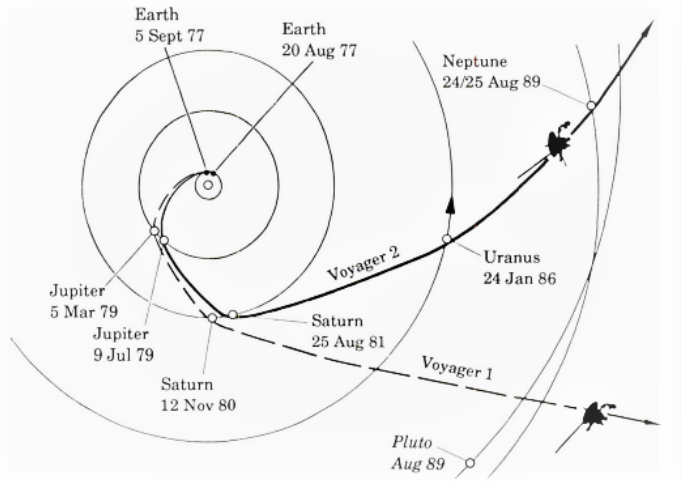


Figure 2.5: Voyager I and II interplanetary trajectories (from [2] Figure 1-4)

The values from those maneuvers are described in tables 2.1 and 2.2, from [21]. The perigee is with respect to the center of the planet.

| | Time | Perigee (km) |
|---|---|---|
| Launch | 5 Sept. 1977 | - |
| Jupiter Flyby | 5 Mar. 1979 | 348890 |
| Saturn Flyby | 12 Nov. 1980 | 184300 |

Table 2.1: Timeline of Voyager I events.

Thanks to the velocity gains obtained during the flybys performed by each probe, they have managed to escape the Solar system and entered interstellar space. Figure 2.6 showcases the evolution of the heliocentric speed of Voyager II.[7]

| | Time | Perigee (km) |
|---|---|---|
| Launch | 20 Aug. 1977 | - |
| Jupiter Flyby | 9 Jul. 1979 | 721670 |
| Saturn Flyby | 26 Aug. 1981 | 161000 |
| Uranus Flyby | 24 Aug. 1986 | 107000 |
| Neptune Flyby | 25 Aug. 1989 | 29240 |

Table 2.2: Timeline of Voyager II events.

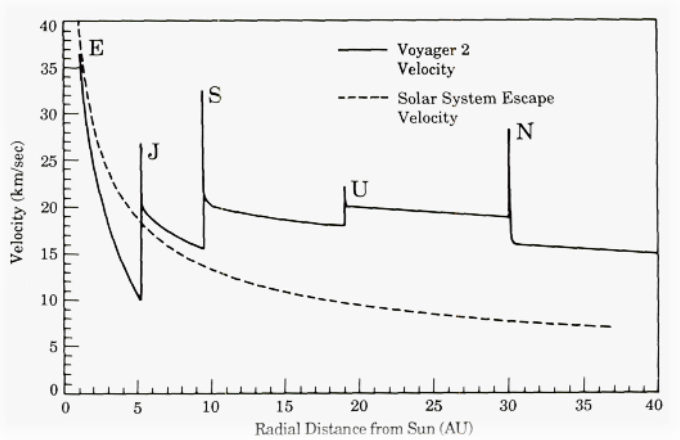


Figure 2.6: Voyager II velocity evolution (from [2] Figure 11-6)

### 2.4.2. Galileo

NASA's Galileo mission launched on the 18th of October 1989 onboard the Space Shuttle. It was aimed at studying Jupiter. In particular, this mission instead of launching to an outer planet, like Mars, it was targeted first at Venus (to an inner planet). The probe performed a first flyby at Venus, followed by two more at Earth before aiming toward Jupiter. While the Voyagers departed directly in the direction of the target planet, Galileo built up speed before departing towards Jupiter. In figure 2.7 we can see the trajectory followed by the spacecraft.[8]

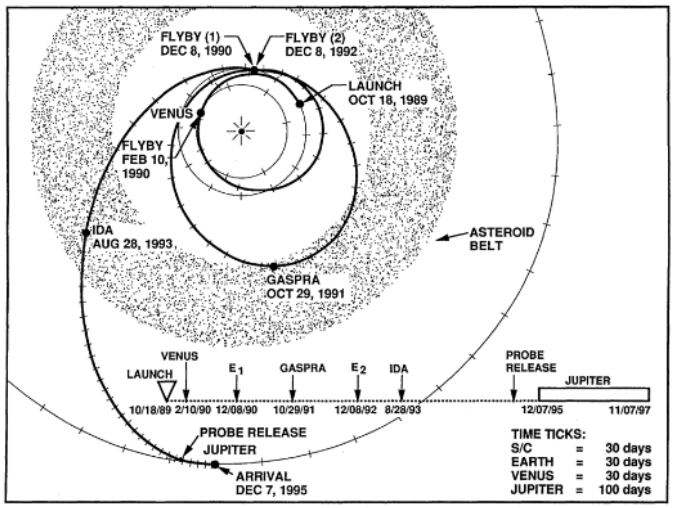


Figure 2.7: Galileo mission interplanetary trajectory (from [3])

The values of the flybys are obtained from [22] and presented in table 2.3.

| | Time | Perigee (km) |
|---|---|---|
| Launch | 18 Oct. 1989 | - |
| Venus Flyby | 10 Feb. 1990 | 22051 |
| Earth Flyby | 8 Dec. 1990 | 7331 |
| Earth Flyby | 8 Dec. 1992 | 6674 |
| Jupiter Arrival | 7 Dec. 1995 | - |

Table 2.3: Timeline of Galileo events.

The mission ended on the 21st of September 2003 when the spacecraft entered Jupiter's atmosphere and disintegrated, after successfully studying the planet and its moons.

### 2.4.3. BepiColombo

BepiColombo is one of the more recent exploration missions that performed several gravity assist maneuvers during its interplanetary trajectory. It's a joint mission from the ESA and JAXA, which was launched on the 20th of October 2018 onboard an Ariane 5. The mission is aimed at studying Mercury. Like Galileo, it performed a double flyby, this time around Venus after the first one at Earth. Figure 2.8 contains the spacecraft trajectory. [9]

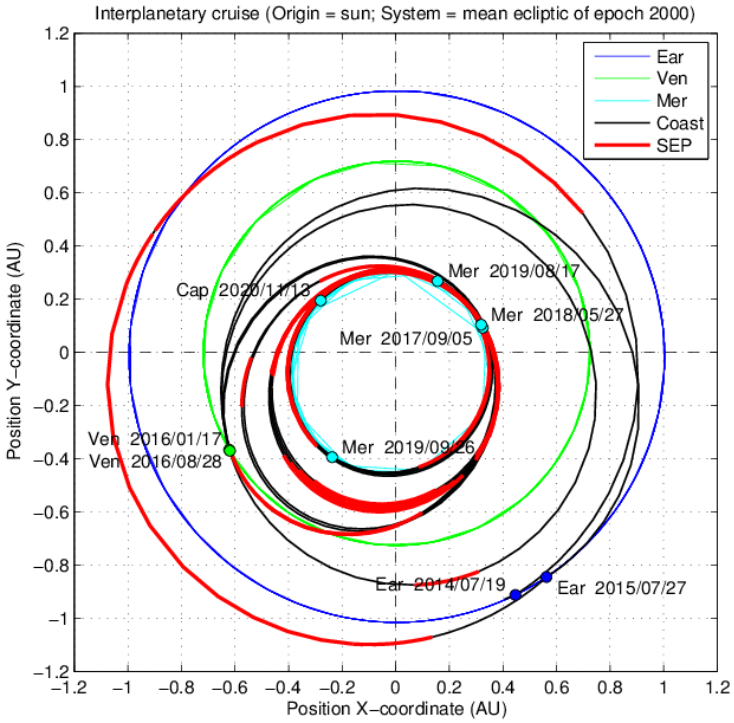


Figure 2.8: BepiColombo mission interplanetary trajectory (from [4])

The spacecraft is currently about the perform the 4th flyby of Mercury in 2023, before the expected end of the mission in 2025.

# CHAPTER 3. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EA) are heuristic optimization methods that operate by mimicking the processes of nature. These algorithms simulate natural selection, cross-over, reproduction, and other natural processes to optimize a problem. This process allows individuals (solutions) in an environment (problem) to better adapt to it, and evolve over time, optimizing, in the process, the solutions to the problem. Although these kinds of algorithms are not deterministic, they can still provide the most optimal solution or at least a near-optimal one.

Evolutionary algorithms were already studied in the 1950s and 1960s by computer scientists [23]. The concept was introduced by Rechenberg [24] and further developed by Schwefel [25]. These algorithms have evolved since their inception, from only mimicking mutation as a means of randomizing possible solutions to a given problem and finding the most optimal, to modeling several natural evolution processes like selection, reproduction, herd movements, ensemble learning, etc. The use of these algorithms has also resulted in the inception of fields such as evolutionary programming, and evolutionary computation.
While evolutionary algorithms are considered artificial intelligence/machine learning (AI/ML) algorithms, they broadly differ in the principal used and their applications when compared with the more traditional AI/ML techniques such as neural networks. These EA rely on a known model of the problem, rather than the use of solutions (data) to model the problem and solve it for new solutions. As a consequence, this kind of methods require previous knowledge of the problem to optimize. However, because the algorithms are agnostic of the problem (the world), but rather the "performance" of a solution to it, they can be easily used for different problems without major changes. This is not the case for neural networks, where new data is required.

EA can be used the optimize problems with high complexity, as long as a solution can be mathematically computed. They are better suited for extensive problems where the search space is really large, and it is impractical for traditional methods to exhaustively search throughout all of it for the optimal solution. EA provide a higher level of abstraction to the problem, simplifying it. However, this comes at the cost of higher computational costs and the convergence to the optimal solution not being guaranteed.

The problem that concerns us fits well within the usage of an EA. The problem has an extremely large search space, and it is hard to model and obtain all the functions required like gradients and mathematical constraints to solve it using deterministic algorithms. The fact that the solution will be optimal or near-optimal is assumed. As we will see in section 5, results from different inputs will output nearly the same solution. While it implies an increase in computational power, a benefit of this kind of algorithm is that some steps of the process can also be parallelized, thus reducing the execution time.

## 3.1. Genetic algorithms

Genetic algorithms (GA) are the most commonly used type of evolutionary algorithms. They were developed in the 1970s by John Holland at the University of Michigan [26]. The research aimed at abstracting the process of natural selection/evolution and employing

these in artificial systems [27]. While GA are simple at their core, meaning, they allow survival of the fittest solution from an original set of random values, they are ameliorated with the introduction of other evolutive processes in the algorithm. This increases performance, robustness, and convergence. This approach is robust in complex search spaces, and even for problems with ill-defined possibilities.

Genetic algorithms are based on the idea of survival of the fittest individual. Each individual is a solution to the problem, from a nature point of view, survivability to the environment. The fittest individual is the one that maximizes a user-defined objective function. The objective function is used as means of evaluating how good a solution is. While from a pure single-objective optimization problem scenario, the fittest one will be the solution that maximizes or minimizes the problem (depending on the objective) from the set of current solutions, other objectives can be defined. As so, GA can also be used to optimize multiple objective problems [28]. For instance, a routing optimization problem, where we want to minimize both the time and the fuel consumption.
In our case, we have a single-objective problem: the minimization of the delta-v required along the interplanetary trajectory (from departure to arrival). Although, the objective function could be made more complex to add more objectives to it. However, this may result in hard-to-judge optimal solutions and can impact negatively the performance of the algorithm. The trajectory's time duration can still be added as a factor in a penalty function.

Individuals are essentially coded as strings of bits denoted "chromosomes". The bits are usually grouped into different portions which make up a "gene". As in nature, the chromosomes are a complete set of genes, where each gene influences a certain part of the individual, hence, the solution. While this representation is a simplification of nature, the main processes can be attained with such representation. Individuals are added as inputs to the problem, which are then evaluated on an objective function basis.

$$\begin{aligned} i : 001011001010 \quad &\longmapsto \quad x \\ y = f(x), & \end{aligned}$$

where $y$ is the fitness and $f$ the objective function.

The individuals are initialized as random values within the search space. This process although simple is crucial to have diversity in the search space. If this process is not correctly done, and diversity is lost, the algorithm will stagnate and will converge to local minima or maxima.
After the initialization process, each individual performance is assessed, i.e. its fitness. The next process involves the use of some genetic operators 3.1.1., to generate new individuals. Several processes of nature are simulated, through which genetic material can be transmitted to new individuals, to allow a convergence/evolution of the population. This process is run several times (each evolution is a generation) until a certain number of generations is reached. A convergence criteria could also be used to end the algorithm, however, the evolution is not bounded to the progression between consecutive runs (convergence) but rather the appearance of random values that increase the fitness of individuals, and as a consequence, the fitness of the future generations.

### 3.1.1. Genetic operators

Genetic operators are the heart of GA, and what makes them differ from purely based random search space algorithms. Genetic operators are used to intelligently select and transmit good genetic information between generations and evolve the population to the optimal solution to the problem.
Let us consider a minimization problem. We have a population, i.e. a group of individuals, represented by chromosomes made up of genes (string of bits). The individuals have already been evaluated and have a fitness score based on their performance.

#### *3.1.1.1. Selection*

The first genetic operator used in a genetic algorithm is selection. As the name suggests, it is used to select the best genetic material to be transmitted to the next generation. In this step, the chromosomes are chosen for reproduction and crossover. The more fit an individual is the more probability of being chosen for reproduction. This operator essentially represents the survival of the fittest individuals and ensures that "good" genetic material is transmitted to the next generations, and not lost.
The two main methods through which this operator is implemented are roulette selection and tournament selection.

1. **Roulette selection:** For this selection method, individuals are singularly chosen based on their fitness. The fitness value rather than maintained as raw is normalized taking into account the whole population. As a consequence, individuals with outstanding performance compared to the rest of the population will be much more likely to be chosen for reproduction. The individuals are sorted based on their normalized fitness, then a random number $r$ between 0 and 1 is generated. The individual selected will be the one that makes the summation of normalized values greater than $r$.

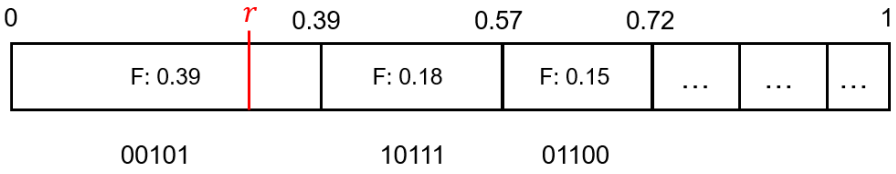


Figure 3.1: Illustration of roulette selection.

Figure 3.1 illustrates the procedure. Individuals are sorted by normalized fitness. The sum of all the normalized fitness is 1. The individual chosen is the one that makes the summation of fitness values including him greater than the random value (the first individual in this example). The better the fitness of an individual, the more "space" it will take in the roulette, and thus, the more probable it is to be chosen.

2. **Tournament selection:** This procedure has an initial selection phase in which several individuals are chosen. In the second phase (the tournament), all the individuals selected are compared and the fittest one is chosen. The selection of the individuals can be done using the roulette selection. In table 3.1, 4 individuals were selected

| Individual | Fitness |
|---|---|
| 01011 | 0.12 |
| 10110 | 0.04 |
| **10001** | **0.23** |
| 10101 | 0.013 |

Table 3.1: Tournament selection method illustration.

using a roulette selection. In the tournament phase, the third individual is the one selected as he has the best fitness.

This operator has the benefit of converging faster the algorithm because more fit individuals are chosen more times. However, it is a greedy operator, it may cause a loss of diversity in the population for the same reason. Hence, leading to a fast convergence, but not necessarily to the optimal value.

#### *3.1.1.2. Elitism*

Elitism is not usually considered part of the genetic operators as it does not affect the creation/evolution of the populations. However, this method is used in cases where the search space is really big. It is used to maintain untouched the best individuals. Those are automatically added to the next generation. Therefore, although these “elite” belonging individuals are considered in the selection and reproduction, they are also added to the next generation with their chromosomes untouched. This is used to prevent the possible loss of the best genetic material due to reproduction, or the individuals not being selected in the selection part.
Similarly to the selection operator, it represents the survival and prevalence of the top-tier individuals. As a consequence, the convergence evolution will always be decreasing or constant in the worst case.

#### *3.1.1.3. Reproduction and Crossover*

Reproduction and crossover operators determine how the genetic material chosen by the selection operator is transmitted to the new generation. It represents the transmission and recombination of good genetic material into new individuals (chromosomes). These operators take two selected individuals and create two offspring (children in nature) out of both chromosomes. The selected parents evolve into two new offspring.
Reproduction is the simplest of the two. It directly takes both parents and forwards them into the next generation, i.e. their chromosomes are left untouched. Crossover on the other hand is the main operator, used to modify the chromosomes of both parents to produce new ones with the genetic material of both. In this case, the parents are crossed, which means that shared genes will remain the same in the offspring, and different genes will be chosen randomly from one of the parents.
There are four main ways in which crossover can be conducted.

- **Uniform crossover:** This first type is the simplest one, yet it performs quite well. It uses the principle of randomly choosing a single bit either from the first or second

parent. If both parents have the same bit, this process is not necessary as the result will be the same. The random selection is done through a coin flip.

Parent 1: 10 110 0111 → Child 1: 11 010 0110
Parent 2: 11 011 0110 → Child 2: 10 111 0111

In this case, there are 4 differences. To obtain the two offspring, we use a coin flip to obtain head or tail, where one means to switch the bits, and the other maintains them. In the example above, we consider heads a switch and tails to maintain the values. The resulting coin flips that we have used are $hhth$.

- **Single gene uniform crossover:** This next method is the same as the previous one, however, the operation is performed on a single gene rather than the whole chromosome. Let us use the same example and consider only the gene in the middle (bit 3 to 5). The head, tails sequence is $th$ (note that in this gene, there are two differences).

  Parent 1: 10 110 0111 → Child 1: 10 111 0111
  Parent 2: 11 011 0110 → Child 2: 11 010 0110

- **Single point crossover:** This method is extremely simple. A random point in the chromosomes is chosen, and the bits between parents are exchanged. Let us consider the random point selected as 3.

  Parent 1: 10 110 0111 → Child 1: 10 111 0110
  Parent 2: 11 011 0110 → Child 2: 11 010 0111

- **Double point crossover:** This last method is the same as the previous one, however, instead of choosing a single point on the chromosomes, two points are selected. Bits are exchanged between those two points. Let us take 3 as the first point and 7 as the second one.

  Parent 1: 10 110 0111 → Child 1: 10 111 0111
  Parent 2: 11 011 0110 → Child 2: 11 010 0110

The operation that two parents undergo (reproduction or crossover) is determined randomly via the use of a random number between 0 and 1. Crossover must have a greater probability of being chosen as it is the operation that creates new genetic material. If single reproduction happens too often, the convergence speed of the algorithm is notably reduced. The majority of genetic algorithms do not consider using reproduction, or reproduction is directly considered as the crossover operation.

Crossover between fit individuals is the main driver of evolution/convergence to the optimal solution. It allows for shared genes of fit individuals to be transmitted to new individuals, and the creation of new ones out of gene differences. This means that “good” genes (as they are present in many fit individuals) prevail in the population, and will eventually become dominant. To exemplify this, let us take a 5-bit chromosome which represents a departure date as a binary number. If all the more fit individuals have the first bit as 1, i.e. the decoded departure date is above 32 ($2^5$), it means that de optimal date is quite likely to be above this number. Therefore, this bit/gene will tend to be dominant, and the population will all have it. The crossover operation allows the transmission of this gene and the determination of the other ones in the same manner.

#### *3.1.1.4. Mutation*

The mutation is the last genetic operator of the evolution process. Although its effects on a single generation are not usually noticeable, in the long run, it has a big effect on the solution. This operator is used to ensure the diversity of the population.
After the reproduction and crossover are done, some random individuals undergo a mutation, i.e. a modification of a part of their genetic material (chromosome). As the population evolves, it tends to lose its diversity and homogenize, decelerating further evolution. This is a consequence of the selection operator, as the fittest individuals are the ones more likely to be selected and their genetic material remains in the population. This genetic material then will extend throughout the population and remain there, because crossing two individuals with the same genetic material, produces the same offspring (no evolution is done). By adding new mutations/changes on the chromosomes, new genetic material can be created, providing more diversity to the population. This operator is also useful when a population becomes stagnated in local maxima or minima.
There are four main types of mutation methods.

- **Flip Bit mutation:** This first method is the simplest. A bit is randomly selected and flipped.

$$11 \quad 011 \quad 0110 \xrightarrow{\text{Bit 3}} 11 \quad 111 \quad 0110$$

- **Boundary mutation:** In this method a sequence of $n$-bits, i.e a single gene is selected and randomly (using a coin flip) flipped completely to 0 or 1.

$$11 \quad 011 \quad 0110 \xrightarrow{\text{Gene 3 to 5}} 11 \quad 000 \quad 0110$$

- **Uniform mutation:** A single gene is selected and the values of the bits that compose it, are each determined by doing a coin flip. It is the same procedure as for the uniform crossover method, however, each coin flips determines the bit value instead of the whole gene. The whole sequence of the gene is new and generated by random numbers (coin flips).

$$11 \quad 011 \quad 0110 \xrightarrow{\text{Gene 3 to 5}} 11 \quad 101 \quad 0110$$

- **Inversion mutation:** For this last method, a gene is selected and inverted.

$$11 \quad 011 \quad 0110 \xrightarrow{\text{Gene 3 to 5}} 11 \quad 110 \quad 0110$$

# CHAPTER 4. ALGORITHM FORMULATION

### 4.0.1. Overview

In the following section, we will present the algorithm that we have developed to optimize the interplanetary trajectory of a spacecraft in the solar system. The complete code will be made public in this repository[1] once the thesis has been presented.
The code/algorithm has been developed from zero without the use of external libraries. It has been mainly coded in C++ and some parts (unrelated to the algorithm's core) for visual purposes in Python. We chose this language as it is one of the most optimal and efficient ones, yet offering the benefits of a standard library. Recall that a genetic algorithm is computationally expensive. Other benefits of using C++ instead of Python or Matlab is that it is compiled, hence faster, and supports multithreading.

The core of the algorithm is divided into two independent parts. The first part is the problem itself, i.e. the design and computation of the interplanetary trajectory. The second part is the genetic algorithm used to optimize the trajectory. While both parts are incorporated into the algorithm, they can be disjointed and used separately, getting back to the fact that a genetic algorithm can be reused independently from the problem to optimize.

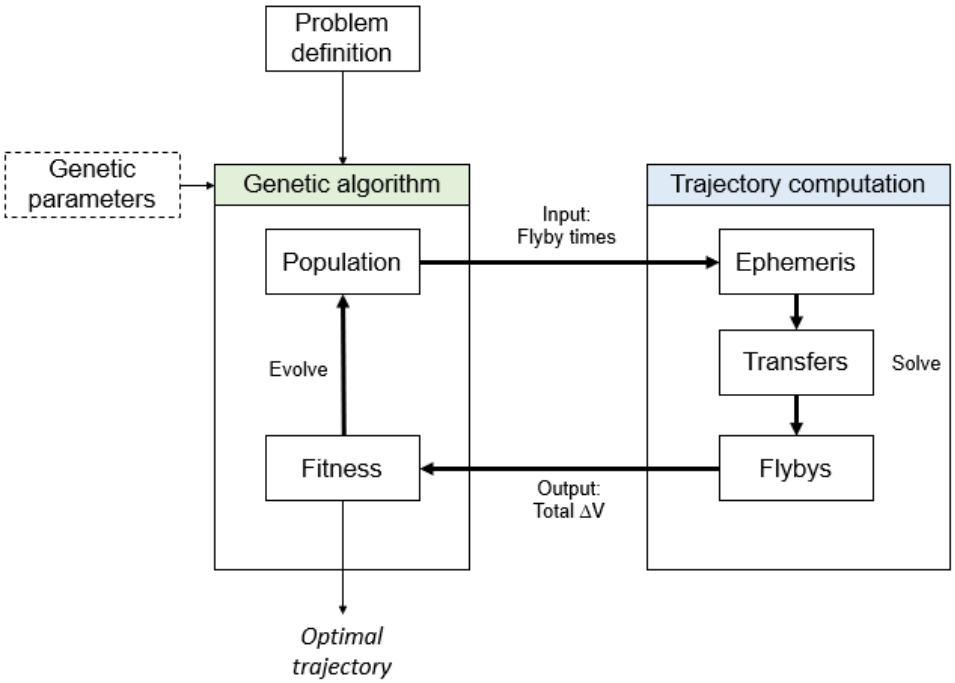


Figure 4.1: Code diagram

Figure 4.1 illustrates the algorithm diagram. The genetic algorithm solves for each individual the trajectory and evaluates the cost of delta-v of the solution obtained. This value is used to evolve the algorithm until a certain number of generations is reached. The global problem is defined at the beginning, together with parameters from the genetic operators

[1] https://github.com/IkerDC/geneticMGA

to modify their behavior, i.e. tweak the algorithm performance. We will see both parts in the following sections.

## 4.1. Trajectory design

The first step in any genetic algorithm is to model the problem. In the context of the GA, the problem represents the world, and the population are the solutions to it. To model it, we need to compute the interplanetary trajectory, provided an input containing the sequence of planets and the times at which each flyby is performed, and obtain a solution, i.e. the required delta-v for that full trajectory.
The simplest way of computing the interplanetary trajectory is by using the patched conic approximation 2.2.. To obtain the trajectory, the problem is broken down into 3 parts. The first part is the computation of the ephemeris of each planet, i.e. at a given time, which is the position and velocity of the planet. Secondly the transfer orbits, between each planet in the sequence, given a departure and arrival date. With this, we compute the two incoming and outgoing hyperbolic velocities to perform such transfers. Finally, the last step is to patch those two hyperbolic velocities using a delta-v maneuver at the perigee of each flyby. In summary, we compute the velocities along all the planned legs of the trajectory, then, patch those velocities together. The action of patching then comes with a delta-v cost. This cost is added to the departure delta-v. The sum is the total delta-v cost of the trajectory.

While this patched conic approach essentially simplifies the problem by approximating it, the solutions obtained are accurate enough to provide a good estimation of the optimal dates and the delta-v cost associated with that trajectory.

### 4.1.1. Ephemeris

The ephemeris of a planet are two state vectors $\vec{R}$ and $\vec{V}$ representing the position and velocity in a heliocentric ecliptic frame of a planet at a given time [1] Sect. 8.10. Those vectors are used to determine the location of the planet at the desired time in the trajectory, as well as its velocity. The heliocentric velocity of the planet is required to compute the flyby maneuver (patched conic).
While the ephemeris are critical, they are computed using analytical data and observation. They were initially based on analytical “theories”, observational data (measurements), and gravitational physics of the solar system [5]. Nowadays, there exist some more complex models, however, they still rely on observation data. This data is obtained using the orbital parameters of the planets and their rate of change. Through them, the state vectors can be computed. This data is only accurate for a certain period, eventually, it needs to be computed again with new observations.
In our case, we will use the table 4.1 of orbital elements from 1850 to 2050, as it provides sufficiently accurate values. For further dates, other tables must be used with less accuracy.

| | $a$ [au, au/Cy] | $e$ [rad, rad/Cy] | $i$ [deg, deg/Cy] | $L$ [deg, deg/Cy] | $w$ [deg, deg/Cy] | $\Omega$ [deg, deg/Cy] |
|---|---|---|---|---|---|---|
| Mercury | 0.38709927 | 0.20563593 | 7.00497902 | 252.25032350 | 77.45779628 | 48.33076593 |
| | 0.00000037 | 0.00001906 | -0.00594749 | 149472.67411175 | 0.16047689 | -0.12534081 |
| Venus | 0.72333566 | 0.00677672 | 3.39467605 | 181.97909950 | 131.60246718 | 76.67984255 |
| | 0.00000390 | -0.00004107 | -0.00078890 | 58517.81538729 | 0.00268329 | -0.27769418 |
| Earth | 1.00000261 | 0.01671123 | -0.00001531 | 100.46457166 | 102.93768193 | 0.0 |
| | 0.00000562 | -0.00004392 | -0.01294668 | 35999.37244981 | 0.32327364 | 0.0 |
| Mars | 1.52371034 | 0.09339410 | 1.84969142 | -4.55343205 | -23.94362959 | 49.55953891 |
| | 0.00001847 | 0.00007882 | -0.00813131 | 19140.30268499 | 0.44441088 | -0.29257343 |
| Jupiter | 5.20288700 | 0.04838624 | 1.30439695 | 34.39644051 | 14.72847983 | 100.47390909 |
| | -0.00011607 | -0.00013253 | -0.00183714 | 3034.74612775 | 0.21252668 | 0.20469106 |
| Saturn | 9.53667594 | 0.05386179 | 2.48599187 | 49.95424423 | 92.59887831 | 113.66242448 |
| | -0.00125060 | -0.00050991 | 0.00193609 | 1222.49362201 | -0.41897216 | -0.28867794 |
| Uranus | 19.18916464 | 0.04725744 | 0.77263783 | 313.23810451 | 170.95427630 | 74.01692503 |
| | -0.00196176 | -0.00004397 | -0.00242939 | 428.48202785 | 0.40805281 | 0.04240589 |
| Neptune | 30.06992276 | 0.00859048 | 1.77004347 | -55.12002969 | 44.96476227 | 131.78422574 |
| | 0.00026291 | 0.00005105 | 0.00035372 | 218.45945325 | -0.32241464 | -0.00508664 |

Table 4.1: Keplerian elements and rates (valid from 1800 - 2500). Table from [5]

In [29] tables for other period ranges can be found.

The six elements in 4.1 are the Keplerian orbital elements of each planet.

- $a$: semi-major axis [au].
- $e$: eccentricity.
- $i$: inclination to elliptic plane [degrees].
- $L$: mean longitude [degrees].
- $w$: longitude of perihelion [degrees].
- $\Omega$: longitude of the ascending node [degrees].

The elements with a dot ($\dot{a}$, $\dot{e}$,...) are the rate of change of each element per Julian century.

Figure 4.2 illustrates how those values are used to represent an orbit.

In order to obtain the state vectors $\vec{R}$ and $\vec{V}$, the following algorithm is used [29]. Dates are expressed in Julian format $JD$. Let us consider a Julian ephemeris date $T_{eph}$.

1. Covert $T_{eph}$ to $JD2000$, i.e. centuries from the year 2000. As

$$T = (T_{eph} - 2451545.0)/36525. \tag{4.1}$$

   Now compute each orbital element at that moment, for instance $a = a_0 + \dot{a}T$, where $a_0$ is the semi-major axis keplerian element (first value) and $\dot{a}$ its rate (second value) on table 4.1.

2. Compute the argument of the perihelion $w$ and the mean anomaly $M$:

$$w = \bar{w} - \Omega \tag{4.2}$$
$$M = L - \bar{w}. \tag{4.3}$$

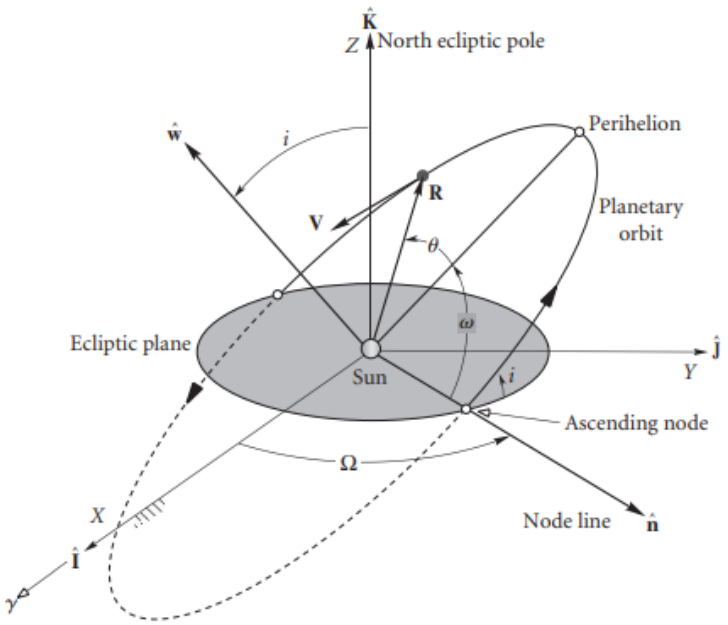


Figure 4.2: Planetary orbit in heliocentric frame reference (from [1] Fig. 8.25)

3. Then, we can numerically solve the Kepler equation (4.4) using the Newton's method.

$$M = E - e\sin E. \tag{4.4}$$

We use an iterative schema with

$$E_{i+1} = E_i - f(E_i)/f'(E_i) \tag{4.5}$$
$$f'(E) = 1 - e\cos E \tag{4.6}$$

In algorithmic form, it can be expressed as follows

$M = E$
**while** $|dE| < 1e-6$ **do**
  $dE = (E - e\sin E - M)/(1 - e\cos E)$
  $E- = dE$
**end while**

4. Then we compute the planet's heliocentric coordinates in its orbital plane, where the $P$-axis points towards the periapsis.

$$P = a(\cos E - e) \tag{4.7}$$
$$Q = a\sqrt{1-e^2}\sin E. \tag{4.8}$$

5. Finally, the coordinates are rotated to a 3D system as follows

$$x_{ecl} = (\cos w\cos\Omega - \sin w\sin\Omega\cos I)P + (-\sin w\cos\Omega - \cos w\sin\Omega\cos I)Q \tag{4.9}$$
$$y_{ecl} = (\cos w\sin\Omega + \sin w\cos\Omega\cos I)P + (-\sin w\sin\Omega + \cos w\cos\Omega\cos I)Q \tag{4.10}$$
$$z_{ecl} = (\sin w\sin I)P + (\cos w\sin I)Q \tag{4.11}$$

With this we obtain $\vec{R}$. Now, in order to compute $\vec{V}$ we can redefine $P$ and $Q$ for the velocity and use the same procedure from that point. We define

$$vP = -(a\sin E \cdot \dot{L})/(1 - e\cos E) \tag{4.12}$$

$$vQ = (a \cdot \cos E\sqrt{1-e^2} \cdot \dot{L})/(1 - e\cos E) \tag{4.13}$$

Then, we rotate the vector like in the last point of the algorithm using $vP$ and $vQ$ in (4.9) to (4.11) instead of $P$ and $Q$.

### 4.1.2. Lambert solver

Once the position $\vec{R}$ and heliocentric velocity $\vec{V}$ of the planets are determined, we need to solve the transfer orbits. To do it, we must solve Lambert's problem.
Lambert's problem was posed by J. H. Lambert in the 18th century. The theorem states that: *The transfer time of a body moving between two points on a conic trajectory is a function only of the sum, of the distances of the two points from the origin of force, the linear distances between the points and the semi-major axis of the conic* [30]. The problem can be stated in functional form as [30] Eq. (3):

$$E = \frac{1}{2}V^2 - \frac{\mu}{r}, \tag{4.14}$$

where $E$ is the energy/mass of the body, $r$ is the distance from the origin of the gravitational force (Sun) to the center of the body, and $V$ is the velocity of the body at $r$. Therefore, if the time of flight $T$ between two points $P_1$ and $P_2$ is determined, then Lambert's problem is to determine the trajectory between both points. The trajectory is determined after finding the departure velocity from point $P_1$ because the position and velocity of any point in the orbit are determined by $r_1$ and $v_1$[1] Sect 5.3. In other words, the next point in the trajectory can be inferred from the previous one. This a consequence of the following equations

$$\vec{r} = f\vec{r_0} + g\vec{v_0} \tag{4.15}$$

$$\vec{v} = \dot{f}\vec{r_0} + \dot{g}\vec{v_0}. \tag{4.16}$$

These equations (2.125 and 2.126 from [1]) describe the concept of obtaining the position and velocity at a moment in time only using the initial values. In (4.15) and (4.16) $f$ and $g$ are the Lagrange coefficients.
Using (4.14) and considering that

$$\begin{aligned} E < 0 &\rightarrow \text{Elliptical conic} \\ E > 0 &\rightarrow \text{Hyperbolic conic} \\ E = 0 &\rightarrow \text{Parabolic conic} \end{aligned}$$

For a hyperbolic transfer orbit between $P_1$ and $P_2$ we can establish a departure speed condition

$$V_1 > \sqrt{\frac{2\mu}{r_1}}. \tag{4.17}$$

This equation is familiar, as it actually is the escape velocity $v_{esc}$.

To solve Lambert's problem, we have used a state-of-the-art open-source solver [31]. The solver has not been implemented as it is quite complex. The solver we have used has been developed in ESA by Dario Izzo and Gooding and it is part of the Pykep[2] project, a scientific library providing basic tools for astrodynamics research.
Given an input of two initial points $r_1$ and $r_2$ and a time of flight $T$, the solver will output the required hyperbolic velocities $v_1$ and $v_2$ to link $r_1$ and $r_2$ in $T$ time. The approach used by this kind of solver is described in more depth in [31], however, it essentially consists of

- The choice of a variable to iterate upon and thus invert the time of flight curve.
- The iteration method.
- The starting guess to use with the iteration method.
- The reconstruction methodology to compute v1 and v2 from the value returned by the iterations.

The code of the solver is available at the Pykep[2] repository and together with our genetic algorithm code.

### 4.1.3. Patched conic

Up to now, we have two out of the three steps required to fully define the interplanetary trajectory. We have the position and velocities of every planet and the trajectory between them. However, they are not joined, each leg is separated. For instance, we may have a trajectory Earth-Mars-Jupiter. Right now, given the ephemeris for a certain date at each planet, we would have the Earth-Mars and Mars-Jupiter legs defined by the departure velocities at Earth and Mars.
To join/complete the trajectory, we must match the arrival velocities from the transfer orbits to the departure velocities (solutions of the Lambert solver). Reusing the previous example, we have to match the arrival velocity at Mars from the Earth transfer to the required velocity departure from Mars to transfer to Jupiter.

To do this operation, we will consider the patched conic approximation. Under the sphere of influence of the correspondent planet, the spacecraft follows a hyperbolic planetocentric trajectory [32]. The result of this matching will be a delta-v difference between both velocities, which would be the necessary delta-v impulse. Recall that this delta-v maneuver will be only allowed at the perigee passage, as at this point is where the impulse would be more efficient (recall the Hohmann transfer). This also allows us to match the incoming and outgoing velocities at the bound of the sphere of influence. Within the sphere, the velocities are set relative to the planet, and thus, only the delta-v and planet velocity relation are considered. The outgoing velocity results from the incoming velocity, plus the flyby delta-v impulse and the velocity provided by the kinetic energy of the planet.

As we have mentioned earlier, we have to match the incoming and outgoing heliocentric velocities provided by the Lambert problem. To match the velocities, we have to find

[2] https://github.com/esa/pykep/

the required turning angle and perigee radius. The difference in velocity will be the delta-v impulse required. We begin by converting the velocities relative to the planet as in [33]

$$\vec{v}_{\infty-in} = \vec{V}_{in} - \vec{V}_p \tag{4.18}$$

$$\vec{v}_{\infty-out} = \vec{V}_{out} - \vec{V}_p, \tag{4.19}$$

where $\vec{v}_\infty$ are the relative velocities, $\vec{V}$ the heliocentric velocities (from Lambert solution) and $\vec{V}_p$ the planet's velocity. Then, we must find the perigee radius of the hyperbolic trajectory to match the two velocities. We start by determining the semi-major axis of both trajectories (at pre and post-perigee passage).

$$a_{in} = -\frac{\mu_p}{v_{\infty-in}^2} \tag{4.20}$$

$$a_{out} = -\frac{\mu_p}{v_{\infty-out}^2} \tag{4.21}$$

where $\mu_p$ is the gravitational parameter of the planet. The turning angle $\delta$ between the asymptote of both trajectories at the bounds of the sphere of influence (Figure 2.4) is computed as

$$\delta = \left( \frac{\vec{v}_{\infty-in} \cdot \vec{v}_{\infty-out}}{v_{\infty-in} \cdot v_{\infty-out}} \right). \tag{4.22}$$

The perigee radius $r_p$ of both parts of the trajectory is the same. It can be expressed as

$$r_p = a_{in}(1 - e_{in}) = a_{out}(1 - e_{out}), \tag{4.23}$$

where $e_{in}$ and $e_{out}$ are the eccentricities of the incoming and outgoing hyperbolic trajectories.
The turning angle can be rewritten in terms of the incoming and outgoing eccentricities as

$$\delta = \sin^{-1}\left(\frac{1}{e_{in}}\right) + \sin^{-1}\left(\frac{1}{e_{out}}\right). \tag{4.24}$$

Then, we can use (4.23) to obtain a function only depending on $e_{out}$. Afterward, we could solve the function for $e_{out}$ using an iterative method like Newton-Raphson as in [33]. However, this approach is unnecessary, if we assume that the trajectory will be correctly joined with the necessary delta-v maneuver at the perigee. We can model the full trajectory as an un-powered flyby from the perigee passage, therefore, we can express the turning angle as in [1] example 8.6

$$\delta = 2\sin^{-1}\left(\frac{1}{e}\right) \tag{4.25}$$

We can isolate the eccentricity from (4.25),

$$e = \sin^{-1}\left(\frac{\delta}{2}\right). \tag{4.26}$$

Then using (4.21) and (4.25) we can solve (4.23).
Finally, the required delta-v applied at perigee is

$$\Delta V = \left| \sqrt{v_{\infty-in}^2 + \frac{2\mu_p}{r_p}} - \sqrt{v_{\infty-out}^2 + \frac{2\mu_p}{r_p}} \right|. \tag{4.27}$$

With these computations, we can essentially patch the two trajectories as long as the delta-v impulse is applied. The perigee radius and the turning angle are determined for the trajectory. We must note that both values are unconstrained, meaning, the only way of patching the two trajectories may involve having an impossible perigee radius or turning angle. For instance, in some cases, the perigee radius may go through the interior of the planet. These cases are considered and avoided later in the algorithm with a penalty function.

At this point, we have all the necessary values to feed the genetic algorithms. For every set of dates defining events in the trajectory, we can compute the full trajectory and obtain a value of how much delta-v that trajectory requires (sum of departure delta-v plus every flyby).

## 4.2. Genetic algorithm

In this section, we will explain in more detail the genetic part of the code 4.1. With the trajectory computation, we can classify solutions with a given fitness based on their delta-v cost and other parameters. Then, evolve the population to converge the solution to the most optimal one.

### 4.2.1. Objective function and constraints

As with every optimization algorithm, we have an objective function. The objective function describes the optimization problem. In our case, the problem and therefore the objective is to minimize the use of delta-v for an interplanetary trajectory with multiple gravity assist maneuvers. While for classical optimization algorithms, the objective function and constraints are separated, meaning, there exist a set of constraints to the problem, but the objective is dissociated from them. In genetic algorithms, constraints cannot be added separately, and are included as part of the objective function through a penalty function as in (1.2)

$$C = f(\vec{X}) + g(\vec{X}), \tag{4.28}$$

where $f$ is the objective (i.e. the delta-v cost) to minimize, and $g$ is a penalty function to incorporate constraints to the solution. $C$ is the objective function that we aim to minimize, which in the genetic algorithm becomes the fitness function for every individual. As we improve the fitness values, we minimize the objective function of the overall problem.

We consider the vector $\vec{X} = [t_0, t_1, \ldots, t_n]$ as the vector of time events in the trajectory; $t_0$ is the departure date, $t_1$ the date of the first flyby, $t_n$ the date of the $n^{th}$ flyby. The function $f(\vec{X})$ is the sum of the departure and flybys delta-v.

$$f(\vec{X}) = \Delta_{departure}(t_0) + \Delta_{fb}(t_1) + \cdots + \Delta_{fb}(t_n). \tag{4.29}$$

The penalty function $g(\vec{X})$ contains two constraints to limit unreal scenarios that can happen during the trajectory computation. Those being a perigee radius that passes through the interior of the planet during a gravity assist maneuver. The second one is to avoid low-velocity flybys between the same planet, i.e. a leg starts and ends on the same planet.

This can result in a solution where the velocity is too low, and the spacecraft gets captured by the planet.
To model the first constraint we initially used [34], where the following function is defined

$$g_i(x_i) = -2\log 10\left(\frac{r_{p,i}}{1.1 r_{r_{pl,i}}}\right), \tag{4.30}$$

where $r_{p,i}$ is the perigee radius at planet $i$ and $r_{pl,i}$ the radius of the planet. We see that the formula will be positive for values of $r_p$ smaller than 1.1 times $r_{pl}$. Therefore, if the perigee radius is not at least 1.1 times the radius of the planet, this penalty function will be positive and thus increase the cost. In other words, it will decrease the fitness of that solution.
However, this approach has not worked in practice as the function is not penalizing enough in most cases. If we solve (4.30) for a perigee radius $r_p$ of 0.9 times $r_pl$ (inside the planet), then $g(x) = 0.17$. This value is not enough to affect sufficiently the objective function. Certainly, such a solution has a greater delta-v reduction, thus cost reduction, than the cost penalty added by such a function. Therefore, to increase the penalty, we decided to use a piece-wise function, where if $r_p > 1.1 r_{pl}$, the penalty cost $g(\mathbf{x})$ is a large constant value

$$g(x) = \begin{cases} 0 & r_p \geq 1.1 r_{pl} \\ 1e10 & r_p < 1.1 r_{pl}. \end{cases}$$

This function is penalizing enough, as the delta-v maneuver usually is in the order of 0 to 10 m/s for the optimal values. Any solution with a perigee radius smaller than $k$-times (usually $k = 1.1$) $r_{pl}$ will be virtually discarded by the genetic algorithms. The $k$ factor is added to prevent the spacecraft from passing too close to the ground and deep within any planetary atmosphere. This would affect the velocity due to the drag, aside from other phenomena that are not modeled in the algorithm.

The second penalty function we have used is from [34].

$$E = \frac{v_{\infty-in}^2}{2} - \frac{G m_{pl}}{r_{SOI}}, \tag{4.31}$$

where

$$r_{SOI} \approx \left(\frac{m_{pl}}{m_{sun}}\right)^{\frac{2}{5}} r_{pl,sun}. \tag{4.32}$$

The penalty function is

$$g(x) = \begin{cases} 0 & E \geq 0 \\ \frac{1}{v_{\infty-in}} & E < 0. \end{cases}$$

This penalty is added to prevent low-velocity flybys, to avoid the spacecraft being captured by the planet's gravity. While we depart with hyperbolic velocity, when we perform a flyby at a planet $i$ followed by another flyby at the same planet after, the velocity required may be close or below to the escape velocity. This is logical, as if the velocity is below the escape velocity of the planet, we would come back to the planet, hence, satisfy the desired trajectory, as we are aiming at a second flyby at this planet. With this penalty function, we make sure this will not happen, and the velocity will always be sufficiently large to avoid the spacecraft being captured.

If any individual in the population has a trajectory where one of these two cases occurs, thanks to the penalty function, its cost will be noticeably high, and it will have a bad fitness value. Therefore, getting virtually discarded from the population, and the possibility of passing their genetic material because there will be an extremely low probability of them being selected by the selection operator for reproduction and crossover.

Other penalty functions could be added. For instance, a time penalty function to prioritize a solution where the trajectory takes less time. However, this will be better added as part of the objective and the genetic algorithm converted to multi-objective optimization. This is mainly because the penalty function should be used for constraints rather than objectives. Hard time constraints are already present as part of the search space limitations, as each time event in (4.29) is constrained to a minimum and maximum time to limit the search space.

$$t_{n,min} \leq t_n \leq t_{n,max}. \tag{4.33}$$

Hence, the duration of the complete trajectory can already be limited. If any soft time constraint is to be added to the penalty function, it would highly affect the optimally of the solution depending on how the penalty is implemented and affects the cost. Therefore, rather than globally aiming at the best solution in terms of delta-v and time, it would aim at the best delta-v solution where time is not penalizing.

### 4.2.2. Genetic evolution structure

Figure 4.3 presents a diagram of how the genetic algorithm is structured.

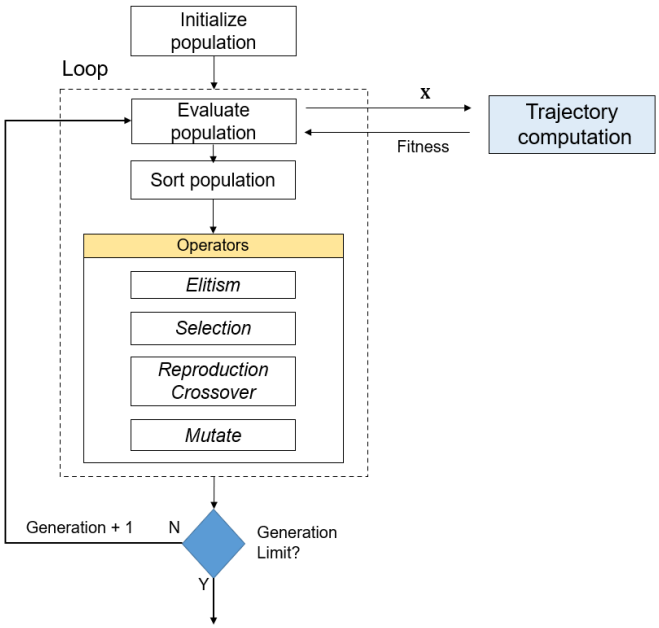


Figure 4.3: Diagram of the genetic algorithm

The genetic algorithm is defined by the size of the population, the number of generations, the method used by each operator, and other probabilities and values used to determine some behaviors of the operators. These values are shared between the algorithm and can

be changed to tweak/modify its behavior. For instance, changing the size and number of generations will have a big impact on the output.
The algorithm begins with the instancing of the population. Each individual is represented by a list of times

$$\vec{X} = [t_0, t_1, \ldots, t_n]. \tag{4.34}$$

where each time has a certain minimum and maximum bound. $t_0$ represents the departure date. The value for each date is cumulative. For instance, $t_0$ is bounded between the following dates: $[2440587.5 \leq t_0 \leq 2441317.5]$, hence, $t_0$ being cumulative can be between 0 and 730 ($2441317.5 - 2440587.5 = 730$). Recall, that the dates are expressed in Julian format. In this last example, those dates correspond to the $1^{th}$ of January 1970 and 1972 respectively. This approach is done to increase the resolution of the input. The same operations are performed for the following dates. Using (4.33) for each $t_n$, the real dates are computed as

$$t_n = t_{0,min} + \sum_{i=0}^{n} t_i. \tag{4.35}$$

The input vector (4.34) is generated randomly. For each date, a random value between the predefined bounds (4.33) is generated.

After this step is performed, we begin the evolution loop. Within the loop, the first step consists of evaluating the performance of the populations. For every individual, the trajectory coded by their dates is evaluated. A fitness value is then derived from their objective function. The next step is to normalize the fitness of the whole population. To do so, we have to consider that we have a minimization problem, therefore, individuals with a lower objective value are considered the more fit individuals. The normalization of the population is done as follows. We begin by computing the adjusted fitness $a(i)$ as

$$a(i) = \frac{1}{1 + s(i)}, \tag{4.36}$$

where $i$ is and individual with a raw fitness $s(i)$. The fitness will now lay between 0 and 1, then, we normalize this fitness for the whole population as

$$n(i) = \frac{a(i)}{\sum_j^n a(j)}. \tag{4.37}$$

After this step, each individual $i$ will have normalized fitness value $n(i)$, and the sum of all fitness values is 1. The fittest individuals will now have larger normalized fitness values.
Ahead of the next step: the operators, the population must be sorted in decreasing normalized fitness order. To do so, we have overloaded a comparison operator in C++ and used the function in the standard library *std::sort(v.begin, v.end)* [35]. This function is optimized to work with large vectors, hence, it is more efficient than using our own implementation.
With the population sorted, we start to apply the genetic operators. Elitism is used first. The first $N$ different individuals of the populations are copied and inserted into a new population vector which will become the next generation. $N$ is one of the parameters that characterize the genetic algorithm, and are defined by the user at the start. Then, we proceed to the selection. We have implemented the two methods presented in 3.1.1.1.. The method to be utilized is part of the input parameters of the GA. The selected individuals are then copied and inserted into the new population vector. Individuals are selected until

the new population vector is full (the size of the vector is the size of the population). Next, the reproduction and crossover operation is performed. This operator acts directly on the new population vector. Two pair consecutive individuals are taken and crossed using one of the methods defined by the user 3.1.1.3.. The two offspring are automatically updated on their parent's positions.
Finally, the last operator, i.e. the mutation is applied. Following the same procedures as before, by using one of the methods 3.1.1.4. selected by the user in the GA parameters, each individual (except the elitism ones) is subjected to a possible mutation. The new population vector then becomes the current population and the cycle is repeated until the maximum number of generations is reached.

While the individuals are coded by a vector of dates, it must be converted to a chromosome, i.e. a string of bits for the reproduction/crossover and mutation operators. Each date $t_n$ in the vector is a gene. The number is converted to binary. The number will take up to 11 bits (2048) for the integral part, and 3 bits for the decimal part. Hence, the possible maximum value of date $t_n$ is 2048.875 days, i.e around 5.60 years. We have considered this value to be large enough for the diverse scenarios and missions with trajectories we want to optimize. The complete chromosome will have a size of $(n+1)*(11+3)$ bits. In C++ this chromosome is a string containing ones and zeros.

### 4.2.3. Multithreading

To improve the execution time of the algorithm we have added multithreading. This procedure allows us to separate independent operations that take a long time to execute into different threads and execute them in parallel at the same time. This procedure can be used in CPUs that allow multithreading, and in programming languages that allow it too; C++ being one of those. On the other hand, for instance, Python does not allow this.
The way we have implemented multithreading is by computing several trajectories (of individuals) in parallel instead of one after the other. While without this improvement, we took each individual and computed their fitness (the trajectory) one by one, now, we have divided the list of individuals into 10 groups, and we solve them at the same time. This is especially useful for large populations. For instance, if we have 25000 individuals, we divided those into groups of 2500 (computed one by one) but solve the 10 groups at the same time.

As we have mentioned, this operation is feasible during the fitness computation because it only depends on the individual parameters. It is an independent operation. During the rest of the genetic algorithm, for instance, the crossing operation, which is relatively expensive and time-consuming, this approach is not possible because the crossing involves dependence between all the individuals. Although, there are other places where multithreading could be added too, such as the mutation operation. Because this operation acts upon single individuals alone, it is also independent and could be done using multithreading. However, we have not considered it because it already is a fast and "cheap" operation, and the performance improvement would be minimal. In addition, there is an increase in code complexity when using multithreading.

This approach is not noticeable for short trajectories where the size of the population is

not so large. For instance, for the Voyager I mission, there is not a meaningful difference in execution time. However, for missions such as Galileo or Voyager II where we have 5 planets in the trajectory (4 legs, 3 flybys), and which involve a larger number of individuals and generations due to the increased search space, multithreading has allowed us to cut the execution time in half. Using a single thread, the execution time for both operations was 7 to 8 minutes, and with multithreading, it is around 4 minutes. As we have mentioned, the time is mostly bounded to the number of individuals and generations, but for such long missions, the sizes of both must be really big to obtain a consistent convergence to an optimum value.

# CHAPTER 5. TEST CASES AND RESULTS

The complete algorithm has been validated in different steps. The trajectory computation has been divided into its three main parts, the ephemeris, Lambert solver, and patched conic computations. The ephemeris and Lambert solver results have been compared and validated against the Pykep[1] library. The algorithms presented here are standard and thus easily verifiable that solution match. The patched conic was compared with data from [33]. In this case, it is harder to validate whether all the parameters computed are the same in both cases. Although the solutions for each part are coherent with the results expected, the computation of the whole trajectory is subjected to the Lambert solver used and the technique used to compute the flybys. For this reason, we must disclaim that whereas the solutions we obtain are the optimal ones for our algorithm, they may differ from the ones obtained by other heuristic or deterministic trajectory optimizers. For instance, the Lambert solver finds a solution to the problem, however, there exist many solutions, and those obtained may differ between different solvers. Therefore, the overall solution will depend on the solver used and the flyby computation technique (in our case we use the patched conic approximation).

Our algorithms have been tested with three of the most relevant missions that followed a MGA trajectory: Voyager-I, Voyager-II, and Galileo. To compare the solutions obtained against the real mission profile, we will use some of the available data of the missions, i.e. the departure and flyby dates, and additionally the perigee radius (available in 2.1, 2.2 and 2.3). This data is quite reduced, and thus hardly representative of the real mission values. To properly compare the results, we will compute the supposed real trajectory values using our model, hence, obtaining the total delta-v, turning angles, etc, of the real missions. Therefore, we will compare two trajectories using the same model, however, for the real design of these missions, a different model was certainly used.

## 5.1. Real missions

### 5.1.1. Voyager-I mission

Let us begin with the Voyager mission. This mission is the one with the simplest trajectory. It departed from Earth in 1977 and performed a single flyby at Jupiter to reach later on Saturn.

In the left table of 5.1 we have the description and input parameters for the genetic algorithm. The population size is quite large, 15000 individuals. This value is high to assure the convergence to the global minima. For this population size, 50 generations are sufficient. These values are quite conservative and could be reduced to decrease the execution time. Of the 15000 individuals, 10 are considered elite, and the rest are selected and reproduced using the tournament and double point techniques respectively. The probability for two individuals to undergo crossover instead of reproduction is 0.9. Later on, with a probability of 0.2, each individual will suffer a mutation using the flip-bit approach.
In terms of the trajectory, we have set large windows to have a sufficiently open search

[1] https://github.com/esa/pykep/

| GA parameters | |
|---|---|
| Population | 15000 |
| Generations | 50 |
| Elitism | 10 |
| Selection | Tournament |
| Crossover | Double Point ($P = 0.9$) |
| Mutation | Flip Bit ($P = 0.2$) |

| Trajectory parameters | |
|---|---|
| $T_{0,min}$ | 01-01-1977<br>2443145 |
| $T_0$ (Earth) | [0, 1095] |
| $T_1$ (Jupiter) | [50, 2000] |
| $T_2$ (Saturn) | [50, 2000] |

| Trajectory Result | | |
|---|---|---|
| | Real | Result |
| **Earth Departure** | | |
| Date $T_0$ | 05-09-1977<br>2443392.5 | 04-09-1977<br>2443391 |
| Departure $\Delta v$ [m/s] | 10330.2 | 9413.34 |
| **Jupiter Flyby** | | |
| Date $T_1$ | 05-03-1979<br>2443937.5 | 17-10-1979<br>2444163.5 |
| $v_{\infty-in}$ [km/s] | 10964.4 | 6558.52 |
| $v_{\infty-out}$ [km/s] | 10960.7 | 6559.38 |
| $\delta$ [deg] | 98.4863 | 93.0884 |
| $r_p$ [m] | $3.376\times10^8$ | $1.1118\times10^9$ |
| $\Delta v$ [m/s] | 1.1371 | 0.34356 |
| **Saturn Arrival** | | |
| Date $T_2$ | 12-11-1980<br>2444555.5 | 24-04-1982<br>2445084.125 |
| Arrival $V_{in}$ [m/s] | 20115.7 | 12704.8 |
| **Result** | | |
| Total cost [$\Delta v$] | **10331.5** | **9414.05** |

Table 5.1: Trajectory optimization for Voyager-I; inputs and solution.

space. The departure window starts the $1^{th}$ of January of 1977 and closes three years later (1095 days). The flyby and arrival windows at Jupiter and Saturn have large windows of 5.33 years each. These windows should be set accordingly to the possible time duration for realistic speeds. In other words, the maximum window time should be higher for greater distances between planets. Otherwise, the speed required to reach the planets in time will be notably high (probably unrealistic). In this case, we may obtain values of $T_1$ and $T_2$ of 50 (which is within the window). In that case, the amount of delta-v needed to transfer in a such small amount of time will be enormous, thus, those trajectories will certainly have really low fitness (really high cost).

The results of the algorithm are described in the right table of 5.1. The mission profile compared with the real one can be observed in figure 5.1. We can see that the optimal trajectory departures at almost the same time (one day of difference). From there on, it takes a slower, more optimal approach. The flyby at Jupiter is conducted 2 years later, in the same year as in the real mission, but 7 months after. The maneuver is done at a higher perigee radius, with a less abrupt turning angle, and a required delta-v impulse of 0.34m/s. The arrival date is April of 1982, compared to the real mission in November of 1980. The total delta-v of the optimal solution is 9.41 km/s, while for the real mission, it is 10.33 km/s.

The difference between both solutions is mainly a cause of the departure delta-v. They both perform a really low delta-v impulse during the flyby, however, an earlier flyby date on the real mission means that the departure speed must be higher to reach Jupiter in time. This implies an extra cost of 0.91 km/s during departure in the actual mission. However, the flyby it performs provides it with more speed, as it reaches Saturn with 20.11 km/s

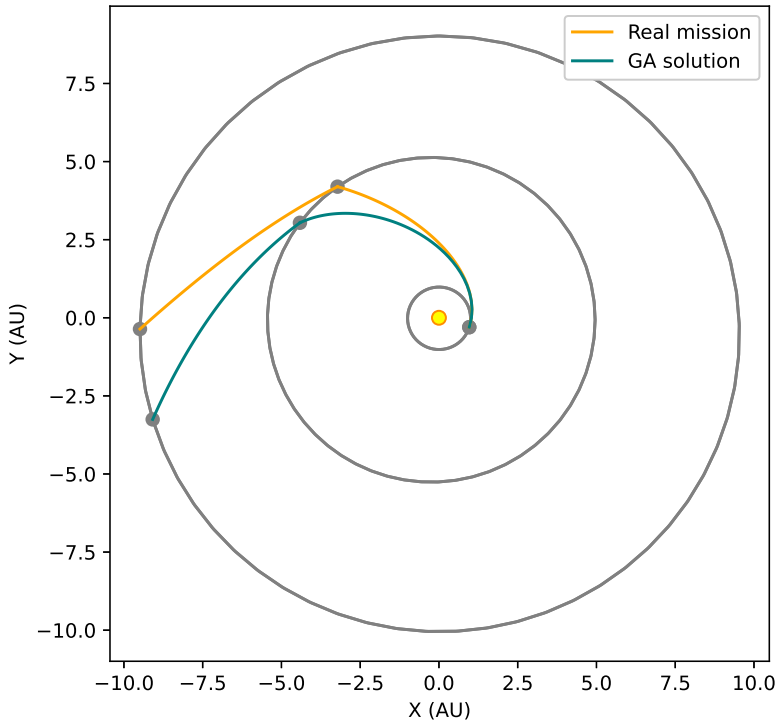


Figure 5.1: Optimal trajectory vs trajectory of Voyager-I.

instead of 12.7 km/s in the optimal trajectory. This is caused by a lower perigee radius, which provides it more speed.
An important observation in Figure 5.1, is that the orbital trajectory is more similar to a Hohmann transfer, which we know is usually the most optimal transfer orbit. In Figure 5.2 we plot the speed evolution of the spacecraft during the trajectory. The sudden spike in speed is the Jupiter flyby.

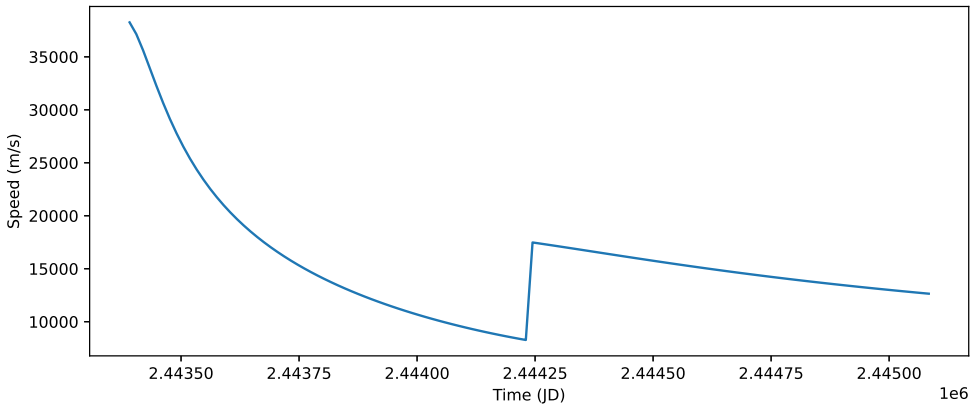


Figure 5.2: Voyager-I optimal trajectory speed evolution.

The net speed gain is around +10 km/s for a maneuver that requires only an injection of 0.00034 km/s from the spacecraft. This gravity-assist maneuver is almost unpowered. The

reason why the algorithm is not providing a solution with a delta-v impulse of 0.0 km/s (unpowered) and only the departure delta-v, is a consequence of the resolution we are using. Times have a maximum precision of 3 hours. We have used 3 bits to define the decimal part of the $JD$ dates, hence, the resolution is $\frac{24h}{2^3} = 3h$. To obtain an unpowered flyby, the time should be more exact.

Thanks to the large population number, the optimal solution is found in early generations. In figure 5.3 we plot the convergence of the algorithms. Around the $10^{th}$ generation, the optimal trajectory is found. This does not mean that evolution stagnates from that point. This means that one of the 15000 individuals in the population has the optimal trajectory coded, however, the population is still quite diversified at this point. Only in later generations, the whole population will tend to this optimal value. Although this evolution strongly depends on how performant the optimal solution is against the rest of the population. The bigger the difference, the earlier its genes will become dominant all around the populations, as more times the individual will be selected.

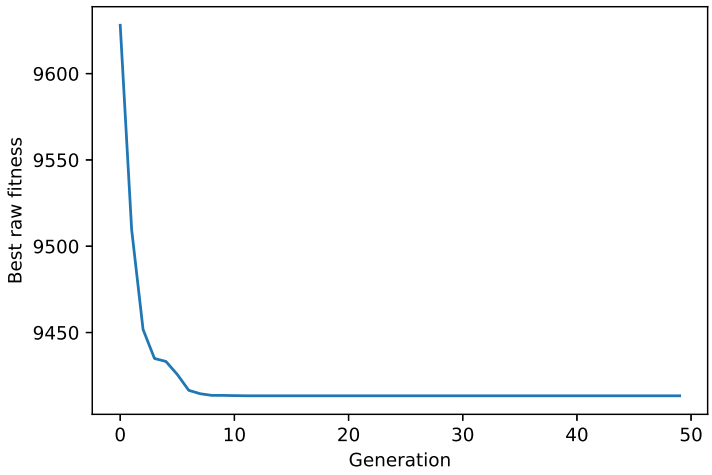


Figure 5.3: Fittest individual evolution for Voyager-I mission.

We must note that in figure 5.3 we plot the fittest individual for every generation. Recall that thanks to the elitism operator, this fittest individual will be preserved in the next generations. Hence, the plot is representative of the fitness of the best individuals, not the fitness and convergence of the whole solution.

### 5.1.2. Voyager-II mission

Voyager-II is the sister mission. It represents an extension of Voyager-I. Instead of stopping at Saturn and being redirected towards deep space, Voyager-II extended its trajectory towards Uranus and Neptune. Due to the mission's larger extension, thus larger search space, the space of solutions is much bigger. This fact must be accounted for in the genetic algorithm.

Table 5.2 presents the parameter we have used in the genetic algorithm. Due to the larger search space, the population size and the number of generations have been increased to

| GA parameters | |
|---|---|
| Population | 50000 |
| Generations | 500 |
| Elitism | 10 |
| Selection | Tournament |
| Crossover | Double Point ($P = 0.9$) |
| Mutation | Flip Bit ($P = 0.2$) |

| Trajectory parameters | |
|---|---|
| $T_{0,min}$ | 2443145<br>01-01-1977 |
| $T_0$ (Earth) | [0, 1095] |
| $T_1$ (Jupiter) | [50, 2000] |
| $T_2$ (Saturn) | [50, 2000] |
| $T_3$ (Uranus) | [500, 2500] |
| $T_4$ (Neptune) | [250, 2500] |

| Trajectory Result | | |
|---|---|---|
| | Real | Result |
| Earth Departure | | |
| Date $T_0$ | 20-08-1977<br>2443375.5 | 04-09-1977<br>2443391 |
| Departure $\Delta v$ [m/s] | 10230.7 | 9413.34 |
| Jupiter Flyby | | |
| Date $T_1$ | 09-07-1979<br>2444063.5 | 17-10-1979<br>2444163.5 |
| $v_{\infty-in}$ [km/s] | 7901.55 | 6563.52 |
| $v_{\infty-out}$ [km/s] | 7757.95 | 6559.38 |
| $\delta$ [deg] | 96.9863 | 93.0884 |
| $r_p$ [m] | $7.058\times10^8$ | $1.1118\times10^9$ |
| $\Delta v$ [m/s] | 57.844 | 0.34356 |
| Saturn Flyby | | |
| Date $T_2$ | 26-08-1981<br>2444842.5 | 24-04-1982<br>2445084.125 |
| $v_{\infty-in}$ [km/s] | 10790.6 | 8254.31 |
| $v_{\infty-out}$ [km/s] | 9052.45 | 8255.15 |
| $\delta$ [deg] | 85.656 | 83.3468 |
| $r_p$ [m] | $2.18\times10^8$ | $2.805\times10^8$ |
| $\Delta v$ [m/s] | 815.66 | 0.378272 |
| Uranus Flyby | | |
| Date $T_3$ | 24-08-1986<br>2446666.5 | 26-05-1987<br>2446942.125 |
| $v_{\infty-in}$ [km/s] | 12877.1 | 11900.2 |
| $v_{\infty-out}$ [km/s] | 17661.6 | 11900.2 |
| $\delta$ [deg] | 22.164 | 18.1143 |
| $r_p$ [m] | $9.21\times10^7$ | $2.583\times10^8$ |
| $\Delta v$ [m/s] | 3728.92 | 0.025 |
| Neptune Arrival | | |
| Date $T_4$ | 25-08-1989<br>2447763.5 | 15-09-1991<br>2448484.125 |
| Arrival $V_{in}$ [m/s] | 21551.0 | 15349.0 |
| Result | | |
| Total cost [$\Delta v$] | **14830.12** | **9414.075** |

Table 5.2: Trajectory optimization for Voyager-II; inputs and solution.

50000 and 500 respectively. A bigger population and generation number imply a larger execution time. However, this is applied in order to assure the convergence to the optimal value and to homogeneously tend to the same value between different runs. The rest of the GA parameters are the same as for Voyager-I. We have maintained the same values as they provide good performance. Different values do not offer a significant change, mostly because the population is sufficiently large that there is always a convergence to the optimal value. The right table in 5.2 presents the results from the genetic algorithm. In figure 5.4 we compare the obtained trajectory and the real one.

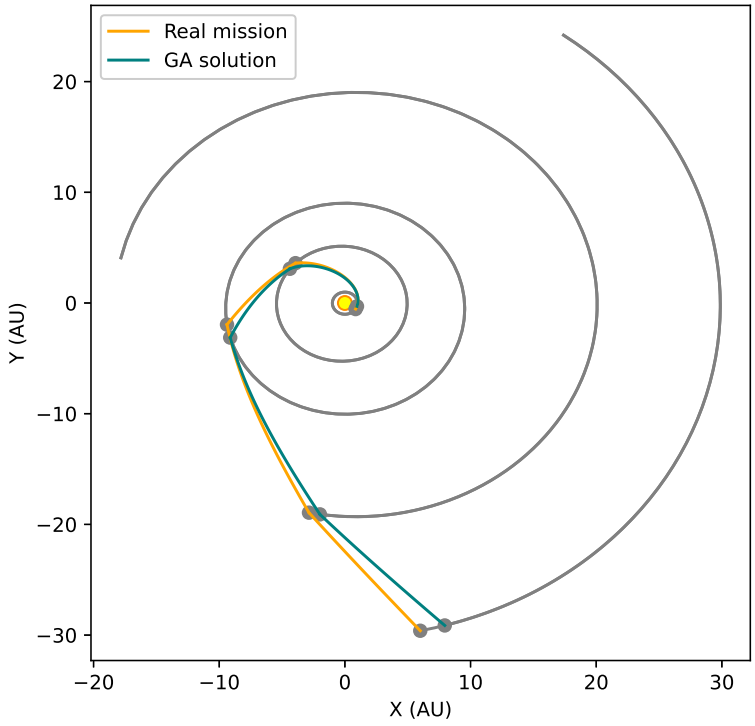


Figure 5.4: Optimal trajectory vs trajectory of Voyager-II.

One notable fact is that the mission until Saturn follows the exact same trajectory and flyby dates as for Voyager-I. For that trajectory, the time sequence is the ideal and the optimal one. For the rest of the trajectory, the algorithm finds the best location of the planets to have flybys as close to unpowered as possible. Hence, obtaining a trajectory where planets in the sequence are carefully located in the ideal locations to gain speed at an extremely low cost.

The Saturn flyby has a delta-v cost of 0.00037 km/s for the optimal solution, compared to 0.815 km/s for the real one, although the speed gained is similar as the perigee radius is similar. During the Uranus flyby, both enter at a similar speed the sphere of influence, however, the real mission flies closer and gains more speed. Although at a higher cost of a 3.782 km/s delta-v impulse. In comparison, the ideal solution has a cost of 0.000025 km/s (almost unpowered). In the end, the real trajectory has a delta-v cost of 14.830 km/s against the 9.14 km/s of the ideal trajectory. Compared with Voyager-I, it only implies a cost of +0.003925 km/s to extend the mission to Uranus and Neptune (with an ideal trajectory).

In terms of mission duration, similar to its sister mission, our solution takes more time; 2 years more. In reality, both missions should depart at the exact same time if their optimal trajectories were to be used.

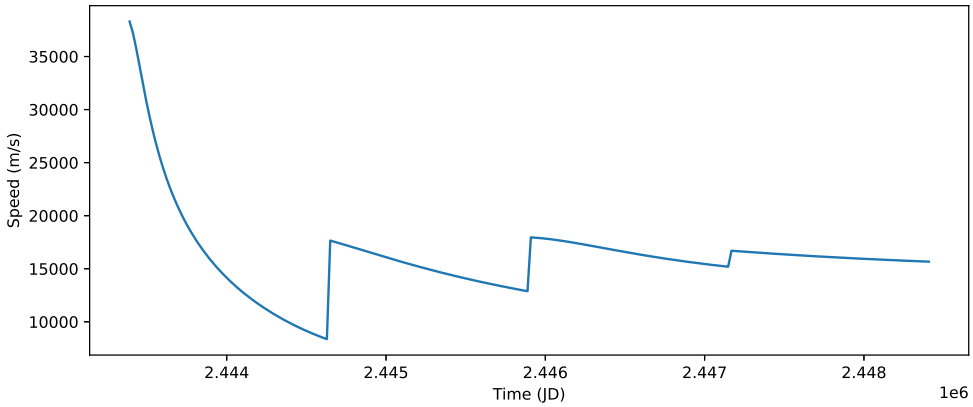


Figure 5.5: Voyager-II optimal trajectory speed evolution.

In figure 5.5 we plot the speed along the complete trajectory of the mission. The speed evolution is quite interesting. The farther the spacecraft is from the Sun, the smaller the speed drop, as the Sun's gravity becomes weaker. Without each gravity assist performed during the mission, not only the spacecraft would have not been able to reach its destination, but neither to have at that point a sufficiently large speed to escape the solar system attraction.

Voyager-I and Voyager-II share a particularity that made them relatively "easy" trajectories. The moment in which they were designed made the trajectory simple because the planets in the sequence were quite aligned. This alignment is measured in time and with respect to the expectable speed of the spacecraft. Although this is not the always the case, the planets were in near-ideal locations at that time to perform gravity-assist maneuvers in a trajectory between them.

### 5.1.3. Galileo mission

The last mission we tested is Galileo. It was launched in 1989, in a more complex trajectory when compared to the Voyagers. It launched from Earth towards Venus (inner planet trajectory) before returning to Earth to do two consecutive flybys, before finally reaching its destination: Jupiter. The parameters of GA are the same as for Voyager-II as the search space has a similar size, they are available at 5.3.

The complete trajectory is showcased in figure 5.6. In this case, the trajectory is harder to interpret. As a consequence, let us use the complete results in the right table of 5.3 to analyze the trajectory. The optimal solution departures 6 months earlier than the real one. It does a similar flyby at Venus with a cost of 0.0071 km/s, before the two consecutive flybys at Earth separated by 1 year and 3 months. Those flybys have an accumulated cost of 0.0352 km/s. We must note, that for the real mission, we have not been able to compute the values for the exact date of the second flyby at Earth. Hence, the *+2* days in the 5.3. We have computed the real trajectory to days later for that flyby because the Lambert

| GA parameters | |
|---|---|
| Population | 50000 |
| Generations | 500 |
| Elitism | 10 |
| Selection | Tournament |
| Crossover | Double Point ($P = 0.9$) |
| Mutation | Flip Bit ($P = 0.2$) |

| Trajectory parameters | |
|---|---|
| $T_{0,min}$ | 2447161.5<br>01-01-1988 |
| $T_0$ (Earth) | [0, 750] |
| $T_1$ (Venus) | [50, 750] |
| $T_2$ (Earth) | [50, 750] |
| $T_3$ (Earth) | [250, 1500] |
| $T_4$ (Jupiter) | [250, 1500] |

| Trajectory Result | | |
|---|---|---|
| | Real | Result |
| **Earth Departure** | | |
| Date $T_0$ | 18-10-1989<br>2447817.5 | 10-04-1989<br>2447627.25 |
| Departure $\Delta v$ [m/s] | 3956.08 | 7329.11 |
| **Venus Flyby** | | |
| Date $T_1$ | 10-02-1990<br>2447932.5 | 12-12-1989<br>2447872.5 |
| $v_{\infty-in}$ [km/s] | 6211.31 | 6165.29 |
| $v_{\infty-out}$ [km/s] | 6222.16 | 6178.85 |
| $\delta$ [deg] | 32.0243 | 68.328 |
| $r_p$ [m] | $2.202\times10^7$ | $6.643\times10^6$ |
| $\Delta v$ [m/s] | 8.166 | 7.182 |
| **Earth Flyby** | | |
| Date $T_2$ | 8-12-1990<br>2448233.5 | 07-02-1990<br>2447930.125 |
| $v_{\infty-in}$ [km/s] | 8960.85 | 9576.54 |
| $v_{\infty-out}$ [km/s] | 5012.07 | 9534.7 |
| $\delta$ [deg] | 100.89 | 43.95 |
| $r_p$ [m] | $4.712\times10^6$ | $7.332\times10^6$ |
| $\Delta v$ [m/s] | 1855.68 | 28.26 |
| **Earth Flyby** | | |
| Date $T_3$ | 8-12-1992 *+2*<br>2448964.5 *+2* | 20-12-1992<br>2448958.505 |
| $v_{\infty-in}$ [km/s] | 5012.93 | 9533.31 |
| $v_{\infty-out}$ [km/s] | 8904.16 | 9541.97 |
| $\delta$ [deg] | 11.83 | 23.9571 |
| $r_p$ [m] | $4.37\times10^7$ | $1.671\times10^7$ |
| $\Delta v$ [m/s] | 3290.23 | 7.0155 |
| **Jupiter Arrival** | | |
| Date $T_4$ | 07-12-1995<br>2450058.5 | 27-03-1996<br>2450170 |
| Arrival $V_{in}$ [m/s] | 7330.42 | 7610.06 |
| **Result** | | |
| Total cost [$\Delta v$] | **9110.45** | **7371.56** |

Table 5.3: Trajectory optimization for Galileo; inputs and solution.

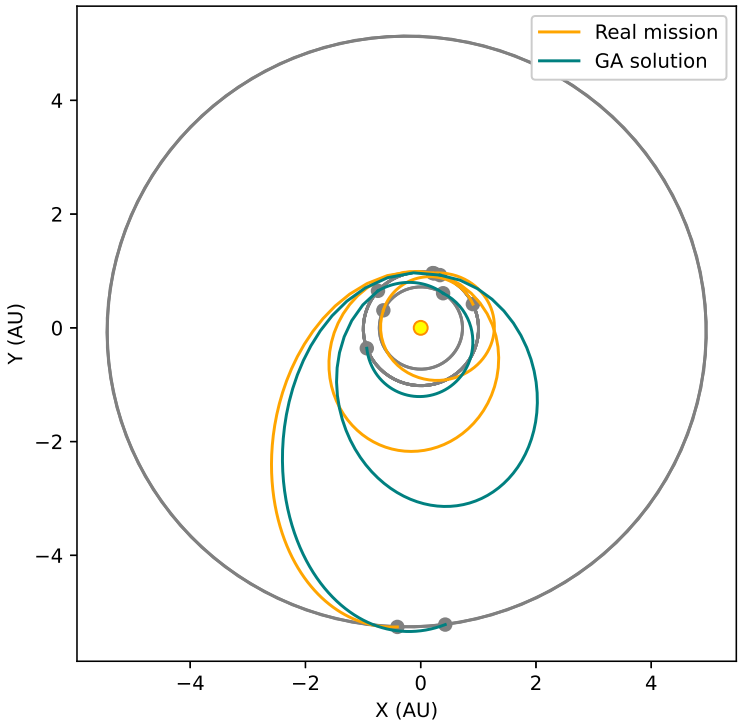


Figure 5.6: Optimal trajectory vs trajectory of Galileo.

solver we used otherwise returned an unrealistic transfer orbit. This is a consequence of the dates being exactly 2 years apart on the same planet. This “unrealistic” solution can be visualized with our code or using the Pykep library for the real dates.
Finally, the total cost of the optimal solution is 7.37 km/s against the 9.11 km/s of the real trajectory. The delta-v required is much lower when compared to the Voyagers because we launch first into an inner planet, instead of an outer one directly. This fact can be illustrated in the speed evolution plot in figure 5.7. The speed increases when the probe is approaching the Sun, and decreases elsewhere, except during the flybys, which are the sudden spikes in the speed.

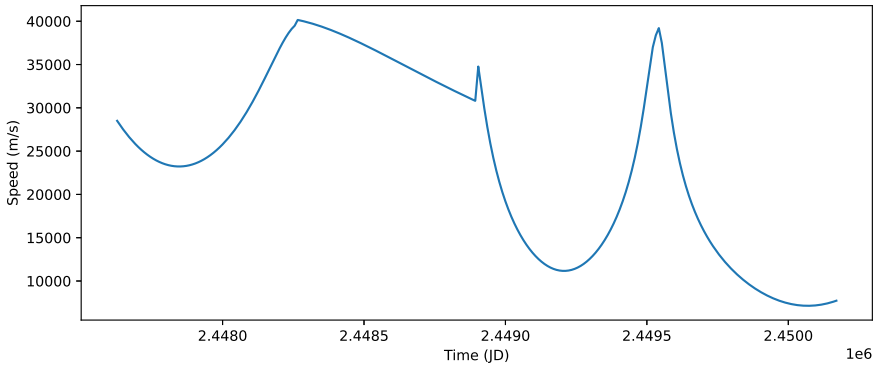


Figure 5.7: Galileo optimal trajectory speed evolution.

## 5.2. Genetic algorithm parameters

### 5.2.1. Population and generations

We have already mentioned that the size of the population should be set depending on the search space size. The larger the space, the bigger the population should be. Along the same line, more generations should be run to allow a proper evolution of the population. In figure 5.8 we take advantage of the fact that the Voyager-I solution are three values (dates) to showcase the deviation (convergence) between 10 different solutions of the algorithm for two different population sizes.

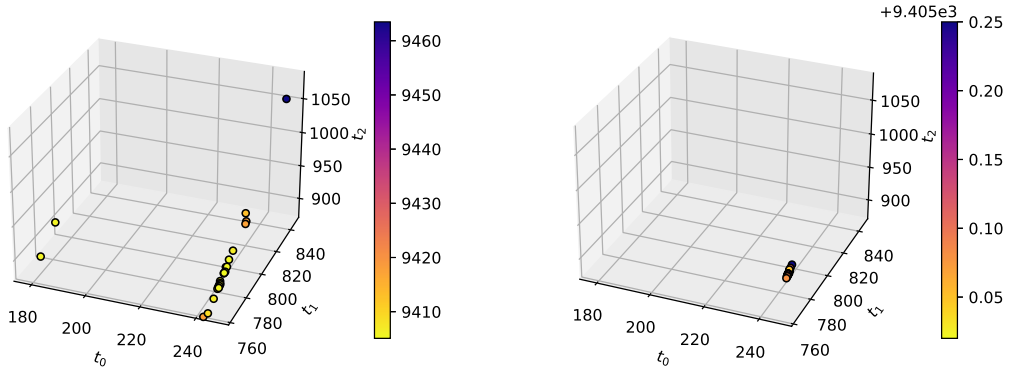


Figure 5.8: Compassion of 10 run solutions: 25 generations with 1500 individuals in the left and 15000 in the right.

We can see that for a sufficiently large population, every run converges to the same minima at the dates specified in 5.1. While if the population size is not large enough, the convergence to the global minima is not assured. In that case, three solutions converged to a local minimum. In that case, those solutions arose in an early generation, and their genes became predominant due to their good fitness when compared to the rest of the population. In that case, the mutation and number of generations were not large enough to still allow for diversity to create a new individual near the actual global minima, and redirect the population towards it.

The population size and generation number have also an impact on the algorithm's performance speed. In Figure 5.9 we plot the convergence of the Voyager-I mission for different sizes of the population.

In this case, we plot the evolution of the best-fit individual for every generation. Recall that this value is not reflective of the convergence of the whole population, but rather of the single best individual of each generation. As the elitism operator is used, the functions are decreasing. We observe that the smaller the population is, the more time it takes the algorithm to converge. Fewer individuals imply a lower probability of finding the optimal value. While for 15000 to 10000 individuals, the optimal values are found around the $10^{th}$ generation, for a population of 500 individuals, the population converges to similar values around the $30^{th}$ generation. Hence, the bigger the population is, the fewer generations are needed to converge the solution.

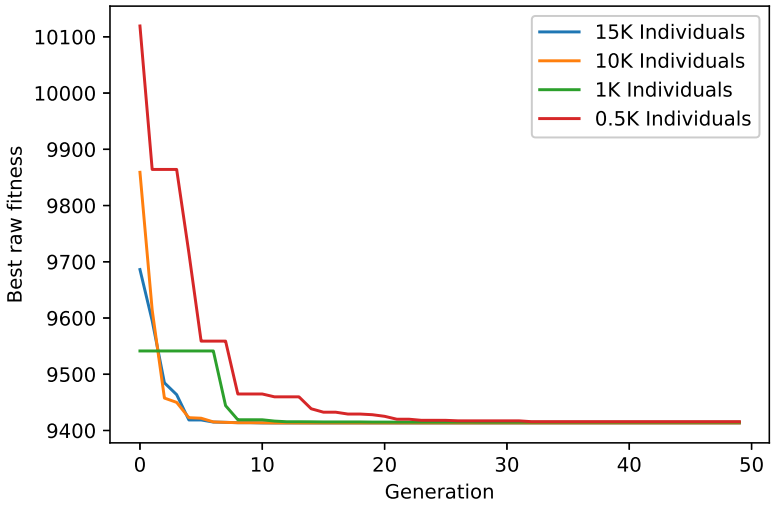


Figure 5.9: Convergence of Voyager-I's mission for different population sizes.

Depending on the problem, one could increase the population size of the generations. Both values have a similar execution time cost. However, increasing the population size should be more effective as there is usually a quicker convergence in early generations.

### 5.2.2. Genetic operators

For the three missions we have used the same genetic operators as they all perform similarly. Because we have used a large population, there is not any apparent effect of changing between different methods of applying each operator.

#### *5.2.2.1. Elitism*

Elitism is an optional parameter that has been set to 10 for every mission. This number corresponds to the number of individuals that are being passed into the next generations directly. The value has low to no effect on such large populations. While a certain quantity of individuals should form part of the elite to avoid losing the best solutions to mutation, the impact on the overall performance is minimum. The operator has a more notable impact on smaller populations in large search spaces, where finding a near-optimal solution is harder. In that scenario, this operator acts as a protection against losing the fittest solution, if it is found.

#### *5.2.2.2. Selection*

Selection operators behave as explained in 3.1.1.1.. The tournament selection is a "greedy" operator. It makes the population converge faster as the fittest individuals are more prone to be selected, but it must be used with more caution. In this case, this method can be used without hesitation because of the large populations. Even if it is a "greedy" method, diversity is not reduced as the number of individuals is really big.

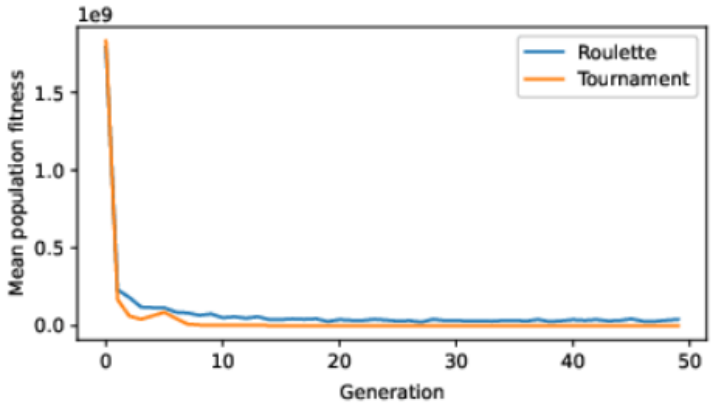


Figure 5.10: Comparison of selection methods for Voyager-I.

Figure 5.10 showcases the difference between both methods for Voyager-I. On it, we consider the mean fitness of the whole population. The tournament operator causes a bigger proliferation of the fittest individual's genes in the population, thus the mean fitness is better (lower trajectory cost).

#### *5.2.2.3. Reproduction/Crossover*

For the reproduction methods, a similar behavior is observed. Below we listed how the algorithm performs for Voyager-I depending method used (same parameters as in 5.1 except the reproduction/crossover).

- **Uniform:** Convergence to optimal value around the $12^{th}$-$15^{th}$ generation. Execution time 4.9s.
- **Single gene:** Convergence to optimal value around the $10^{th}$ generation. Execution time 3.3s.
- **Single point:** Convergence to optimal value around the $10^{th}$ generation. Execution time 4.4s.
- **Double point:** Convergence to optimal value around the $10^{th}$ generation. Execution time 4.3s.

Each method converges to the same solutions. The execution time is the fastest for the single gene crossover methods. This method is faster as the operation to perform it is much quicker. The other operations involve converting the chromosomes to a bit-string and operating upon it. On the other hand, single gene crossover only involves exchanging two numbers. For the convergence speed, the uniform method is slightly slower.

In general, the methods used to implement the genetic operator do not have a big impact on this optimization problem. However, the population size and number of generations are more critical and will affect the outcome. In order to have a proper and coherent convergence to the optimal values, both values should be big, although at the cost of a longer execution time.

# CHAPTER 6. CONCLUSION

In this thesis, we aimed to investigate and develop a non-traditional approach to solve the optimization problem of multi-gravity assist (MGA) interplanetary trajectories. It's not long ago, that computer technology only allowed us to solve this kind of problem by properly defining and modeling the problem. These highly complex modeling of functions, gradients, and singularities, could be used with traditional optimization methods. Nowadays, the current technology has allowed the use of new kinds of algorithms to solve such optimization problems. In this thesis, we have demonstrated the use of evolutionary algorithms, especially, *genetic algorithms* to optimize interplanetary trajectories. A genetic algorithm has been coded from the ground up together with the different computations required to model the full trajectory. These trajectories have been modeled using a Lambert solver and the patched conic approximation.

Through the validation and testing of the algorithm against real missions such as Voyager I and II, and Galileo (which followed an MGA trajectory), we have demonstrated that this approach can find near-optimal solutions. The algorithm's convergence to the optimal value can reliably be obtained by correctly selecting the different parameters of the genetic algorithm. One notable finding in that regard is that the implementation of the genetic operators does not have a meaningful impact on the performance of the algorithm. While these genetic operators are the heart of the genetic algorithm, the different methods used on them will only slightly affect the convergence speed and execution time of the algorithm. The final solution is less dependent on the method used. On the other hand, we have shown that the size of the population and the number of generations will highly impact the outcome of the algorithms. The sizes of both should be large enough to accommodate the search space. The greater the search space is, the larger these values have to be. Otherwise, the algorithm may not converge to the optimal value (global minimum) and instead converge to a local minimum. As well, the solutions obtained in different runs may differ to a greater extent, as this convergence cannot be assured.

Genetic algorithms are simplistic, meaning, they only require an objective function. In an optimization context, this objective function is the function to optimize. In our case, it has been the delta-v cost of the trajectory, as this value is more representative of how optimal an interplanetary trajectory is. With this objective function and some constraints, the problem can be modeled. The simplicity aspect has a trade-off in terms of computational power and execution time which are larger than in traditional methods. However, as discussed earlier, current technology is able to handle these requirements. In addition, the code performance can be improved using techniques like multithreading as we have showcased for the trajectories computations.

In a summary, *genetic algorithms* have been shown as a viable alternative to traditional methods to find near-optimal MGA interplanetary trajectories. They are much more simple and easier to implement in exchange for more computational power. The solutions provided by the algorithm match the expected results and convergence criteria, although these solutions are highly dependent on the model used to model the trajectory.

## 6.1. Future work

The are two main lines that can be followed as future work to the thesis. The first one is along the lines of improving the cost of the genetic algorithm. As we have discussed in the thesis and has been showcased during the execution of the GA, this kind of method is computationally expensive and time-consuming. For instance, in a modern CPU, to be exact a *Intel Core™ i7-10510U* with 8 GB of RAM, running Linux, it took in the range of 4-5 seconds to execute the code for Voyager-I. Recall that this mission has a relatively small number of individuals and generations (15000 and 50). For Voyager-II and Galileo, the execution took around 7-8 minutes (without multithreading).
As an extension to the current work, the GA could be improved to reduce the exception time. Some optimizations could be added during the compilation (-03 g++ flag[1]), which may drastically reduce the execution time at the cost of floating precision. This is not ideal as these optimizations come with a precision loss. For the problem that concerns us, large values are used, thus precision is key to obtaining a correct result. Another option that has been used in the code, and could be developed further, is to use multithreading. By computing consecutive and independent operations in parallel, the time can be notably reduced. In our case, multithreading has been implemented partially to compute the trajectory of every individual in groups of 10 in parallel during run-time. However, there are other parts in the code where this technique can be used, for instance, the selection of individuals and mutation. The number of threads utilized could be optimized to improve the time performance of the algorithm.
In the same line, another option studied in [33] will be to use the GPU to increase the computational power available. As the GPU is devoted to graphical operations, which are usually more complex and expensive than classical operations handled by the CPU, it has greater computational power. The algorithm could be implemented using the CUDA[2] or OpenCL[3] APIs to use the GPU processing unit. It must be noted that such APIs are complex and low-level. They are normally implemented as part of graphical engines. These engines provide the user with a high-level implementation, while in the interior they use these lower-level APIs. For our genetic algorithm, this would not be the case. However, using the GPU can improve notably the execution time.
This line of research could allow us to optimize much larger and more complex MGA trajectories in less time. It would as well open the door to using more complex models to compute/model the trajectory, instead of approximations as the patched conic. Hence, obtaining a more precise solution.

The second line of research lies in increasing the capabilities of the algorithm. In our case, the algorithm is constrained to the MGA model, instead of the MGA-DSM model. This last one allows for single or multiple delta-v impulses during any leg of the trajectory. It is not constrained to the impulse being performed at the perigee of a flyby. Implementing this model would make the algorithm more versatile and allow the optimization of missions that use these deep-space maneuvers. For instance, the Cassini mission [16] used this kind of trajectory.
The algorithm could also be more flexible by implementing a parallel optimization to com-

[1] https://gcc.gnu.org/onlinedocs/gcc-8.1.0/gcc/Optimize-Options.html#Optimize-Options
[2] https://developer.nvidia.com/gpu-accelerated-libraries
[3] https://www.khronos.org/opencl/

pute the ideal planet sequence as in [34]. Through this, the planet sequence could be unspecified by the user, and instead, optimized by the algorithm in conjunction with the complete trajectory. This approach would work for missions defined by a departure and final destination planet. This approach would avoid potentially pruning out the optimal trajectory due to planet sequence being prefixed.
In general, this line would increase the flexibility of the algorithm when it comes to optimizing different types of trajectories. In addition, it would allow it to solve more complex missions, and provide more complex (and optimal) trajectories.

# BIBLIOGRAPHY


[1] H. Curtis. *Orbital Mechanics: For Engineering Students*. Aerospace Engineering. Elsevier Science, 2015. ix, 5, 6, 7, 8, 9, 10, 22, 24, 25, 27

[2] C. Kohlhase. *The Voyager Neptune travel guide.* 1989. ix, 11, 12

[3] Christopher T Russell. *The Galileo Mission*. Springer Science & Business Media, 2012. ix, 12

[4] Rüdiger Jehn, Johannes Schoenmaekers, Daniel García Yárnoz, and Paolo Ferri. Bepicolombo-a mission to mercury. 09 2009. ix, 13

[5] Myles Standish and James Williams. *Orbital Ephemerides of the Sun, Moon, and Planets*. 01 2006. xi, 22, 23

[6] Donna L. Shirley. The mariner 10 mission to venus and mercury. *Acta Astronautica*, 53(4):375–385, 2003. 1

[7] Charles Kohlhase and Paul Anthony Penzo. Voyager mission description. *Space Science Reviews*, 21:77–101, 1977. 1, 11

[8] Project galileo at jupiter. *Acta Astronautica*, 40(2):477–509, 1997. 1, 12

[9] Alberto Anselmi and George E. N. Scoon. BepiColombo, ESA's Mercury Cornerstone mission. , 49(14-15):1409–1420, December 2001. 1, 13

[10] Yanping Guo and Robert W. Farquhar. New horizons mission design for the pluto-kuiper belt mission. 2002. 1

[11] Steve Matousek. The juno new frontiers mission. *Acta Astronautica*, 61:932–939, 11 2007. 1

[12] Titan 3e/centaur d-1t systems summary. *NTRS - NASA Technical Reports Server*, 1973. 1

[13] A.A. Siddiqi and United States. NASA History Program Office. *Beyond Earth: A Chronicle of Deep Space Exploration, 1958-2016*. NASA SP. National Aeronautics and Space Administration, Office of Communications, NASA History Division, 2018. 1

[14] Kaijian Zhu, Junfeng Li, and Hexi Baoyin. Multi-swingby optimization of mission to Saturn using global optimization algorithms. *Acta Mechanica Sinica*, 25(6):839–845, December 2009. 2

[15] Matteo Ceriotti. Global optimisation of multiple gravity assist trajectories. 2010. 2

[16] Linda Spilker. Cassini-huygens; exploration of the saturn system: 13 years of discovery. *Science*, 364(6445):1046–1051, 2019. 2, 48

[17] D.R. Myatt S.J.Nasuto J.M.Bishop D. Izzo, V.M. Becerra. Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories. 3

[18] John W. Hartmann, Victoria L. Coverstone-Carroll, and Steven N. Williams. Optimal Interplanetary Spacecraft Trajectories via a Pareto Genetic Algorithm. *Journal of the Astronautical Sciences*, 46(3):267–282, September 1998. 3

[19] John Prussing. Simple proof of the global optimality of the hohmann transfer. *Journal of Guidance, Control, and Dynamics*, 15, 09 1992. 5

[20] Edward Belbruno and John Carrico. Calculation of weak stability boundary ballistic lunar transfer trajectories. 08 2000. 7

[21] PDS: The Planetary Data System. Mission information, 2017. 11

[22] NASA. Galileo mission in-depth, 2017. 13

[23] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 03 1998. 15

[24] Ingo Rechenberg. *Bionik, evolution und Optimierung*. Wissenschaftliche Verlagsgesellschaft mbH, 1973. 15

[25] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie*, volume 1. Springer, 1977. 15

[26] John H. Holland. Adaptation in natural and artificial systems. 1975. 15

[27] David E. Goldberg. Genetic algorithms in search optimization and machine learning. 1988. 16

[28] Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering System Safety*, 91(9):992–1007, 2006. Special Issue - Genetic Algorithms and Reliability. 16

[29] NASA JPL. Approximate positions of the planets. 23

[30] James F. Jordon. The application of lambert's theorem to the solution of interplanetary transfer problems. Technical report, NASA - Jet Propulsion Lab, 1964. 25

[31] Dario Izzo. Revisiting lambert's problem. *Celestial Mechanics and Dynamical Astronomy*, 03 2014. 26

[32] Victor R Bond. *Matched-conic solutions to round-trip interplanetary trajectory problems that insure state-vector continuity at all boundaries*, volume 4342. National Aeronautics and Space Administration, 1969. 26

[33] Sam Wagner, Brian Kaplinger, and Bong Wie. Gpu accelerated genetic algorithm for multiple gravity-assist and impulsive v maneuvers. 08 2012. 27, 35, 48

[34] Jacob A. Englander, Bruce A. Conway, and Trevor Williams. Automated mission planning via evolutionary algorithms. *Journal of Guidance, Control, and Dynamics*, 35(6):1878–1887, 2012. 29, 49

[35] cplusplus.com. C++ sort algorithm documentation. 31