# Global Trajectory Optimisation: Can we Prune the Solution Space when Considering Deep Space Manoeuvres?

## Final Report

**Authors:** Massimiliano Vasile, Matteo Ceriotti, Gianmarco Radice
**Affiliation:** Department of Aerospace Engineering, University of Glasgow, UK
**Authors:** Victor Becerra, Slawomir Nasuto, James Anderson
**Affiliation:** School of Systems Engineering, The University of Reading, UK

**ESA Researcher(s):** Claudio Bombardelli

**Date:** 01/01/2008

**Contacts:**

Massimiliano Vasile
Tel: +44 (0)141 330 6465
Fax: +44 (0)141 330 5560
e-mail: m.vasile@aero.gla.ac.uk

Claudio Bombardelli
Tel: +31 (0)71 565 8718
Fax: +31 (0)71 565 8018
e-mail: act@esa.int

Available on the ACT website
http://www.esa.int/act

**Ariadna ID:** 06/4101
**Study Duration:** 6 months
**Contract Number:** 20273/06/NL/HE

# TABLE OF CONTENTS

*(This page is left intentionally blank)*

# PREFACE

This document contains a report on the work done under the ESA/Ariadna study 06/4101 on the global optimization of space trajectories with multiple gravity assist (GA) and deep space manoeuvres (DSM). The study was performed by a joint team of scientists from the University of Reading and the University of Glasgow.

## General introduction

Multiple gravity-assist trajectories have been extensively investigated over the last forty years, and their preliminary design has been approached mainly relying on the experience of mission analysts and following simplifying assumptions in unison with systematic searches or some simple analysis tools such as the Tisserand's graph [1, 2].
On the other hand, since at the early stage of the design of a space mission a number of different options is generally required, it would be desirable to automatically generate many optimal or nearly optimal solutions over the range of the design parameters (escape velocity, launch date, time of flight, etc…), accurately enough to allow a correct trade-off analysis.
Recently, different attempts have been carried out toward the definition of automatic design tools, although so far most of these tools have been based on systematic search engines. An example is represented by the automatic tool for the investigation of multiple gravity-assist transfers, called STOUR, originally developed by JPL and subsequently enhanced by Longuski et al. at Purdue university [3]. This tool has been extensively used for the preliminary investigation of interplanetary trajectories to Jupiter and Pluto [4], for the design of the tour of Jovian moons and for Earth-Mars cycling trajectories.
In the last ten years, different forms of stochastic search methods have also been applied to orbit design, starting from the work of Coverstone et al. [5] on the use of multi-objective genetic algorithms for the generation of first guess solutions for low-thrust trajectories, to more recent works on the use of single-objective genetic algorithms for ballistic transfers [6] or to the use of hybrid evolutionary search method for preliminary design of weak stability boundaries (WSB) and interplanetary transfers.
More recently, it has been shown [7, 8] that if powered swing-bys are considered and no deep-space manoeuvre are introduced, the solution space of multiple gravity assist optimisation problem (MGA) can be pruned considerably in polynomial time (with a small exponent). This particular property allows an efficient solution of even highly complex trajectories in polynomial time through a deterministic branch and prune algorithm.
However, an MGA trajectory model with no deep-space manoeuvres does not allow to design a number of interesting trajectories. If the problem of multiple gravity assist with deep space manoeuvres (MGADSM) could be pruned in polynomial time with a small exponent, an efficient branch and prune algorithm could be used as in the simpler MGA case. The introduction of DSM offers the further advantage of providing a reasonable

approximation of MGA trajectories with low-thrust arcs, thus allowing to generate first guess solutions also for that kind of trajectories.
If a transfer arc is no more simply ballistic but is shaped by one or more propelled manoeuvres (either impulsive or low-thrust) the number of degrees of freedom increases significantly. Hence, an efficient deterministic solution process would have to make use of additional information (with respect to the simple ballistic case) to cut down the number of possible alternatives. Moreover a complete automatic tool should allow to select and combine the most appropriate transfer arcs (ballistic, deep-space, low-thrust, as for example in the JPL code STOUR-LTGA where exponential sinusoids are used together with ballistic arcs [7]). Finally the optimal sequence of celestial bodies should be selected automatically since, as demonstrated in [7], its correct choice has a major impact on the final result.
Few examples of free sequence solutions exist, some using a deterministic two level approach in which the sequence is selected at an upper level and then is optimised at a lower level [3, 8], few other use an integrated approach in which discrete and continuous quantities are treated together in the same formulation [9, 10].
The most general case in which the model would contain integer, real and logical quantities (for the selection of the transfer arcs) all together can be formulated as a hybrid optimisation problem. The solution of these kinds of problems is still an open issue and few recent examples can be found in the literature [11, 12]. In these examples hybrid optimal control problems, which are a special case of hybrid system theory, are solved by a combination of direct collocation, for continuous variables, and of some branch and bound or integer programming approach to deal with discrete quantities.
Hybrid optimal control problems are conceptually equivalent to the problem we are addressing in this study and similar solution techniques can be applied even to our case.
In this study we will address the solution of hybrid problem by a combination of deterministic and stochastic techniques.

# Study objectives

In order to address the definition and implementation of an efficient tool for the solution of the MGADSM problems, the present study aims at reaching the following objectives.
The primary objective of the study should be to expand the results obtained on the multiple gravity assist problem [13, 14] to the more complex case in which one or more trajectory legs include a deep space manoeuvre. In particular:

**1. Pruning the solution space in case one or more deep space manoeuvres are included in the trajectory description**

The algorithm complexity for the problem of designing a trajectory made of multiple gravity assist manoeuvres and deep space manoeuvres should be analysed. The outcome of this analysis should lead to the definition of a set of efficient pruning techniques for this specific problem. The proposed pruning techniques should allow a fast solution of problems with a large number of deep space manoeuvres. The search space reduction obtained with the developed pruning techniques should allow a reliable identification of: launch windows, mission options over a wide range of launch dates, mission cost in terms of $\Delta v$, epoch of the manoeuvres for a given sequence. The pruning technique can be combined with additional search mechanisms that operate on the remaining parts of

the solution space. This part of the work will require the development of a trajectory model that includes deep space and gravity assist manoeuvres. The complexity analysis will be performed on the model developed and the pruning technique will make use of the characteristics of the model.

**2. Integration between the various blocks comprising the trajectory**

A general trajectory has to be described in terms of elementary building blocks, such as: purely ballistic arcs, ballistic arcs with one or more deep space manoeuvres, low-thrust arcs, gravity assist manoeuvres, departure and arrival conditions. The optimal combination of the building blocks and the simultaneous optimisation of each single one can be formulated as a hybrid optimisation problem with discrete and continuous quantities. Elements such as deep space and gravity manoeuvres can be seen as singular events. The hybrid problem should allow the selection of number and sequence of legs, as well as number and sequence of singular events.

**3. Feasibility proof of such integration for a number of agreed test cases**

The search and pruning methodology applied to the integrated hybrid problem should be tested on a number of selected cases. The relevant indexes of performance and metrics will be the time needed to identify a set of solutions and the number of solutions in the set, the feasibility and local optimality of the solutions (distance from the local minimum, where the local minimum is located by using a local optimiser initialised at the solutions found by the hybrid optimiser), optimality of the generated solutions in comparison to other techniques.

**4. Feasibility proof of the use of PSO for global trajectory design**

A secondary study objective is to apply Particle Swarm Optimisation (PSO) [15] to the pruned part of the solution space. For this part of the study it is required to investigate how to automatically tune the relevant parameters of PSO to make its performance adaptable to a wide range of problems.

# Document structure

The document is structured into three main parts:

- PART 1: MODELLING. This part contains an extensive description of all the mathematical and computational trajectory models that were used in the study. This part includes a description of each of the components of a trajectory model and a discussion on the expected computational complexity of the algorithms. In particular two trajectory models will be presented: one with an exact a priori satisfaction of the physical constraints characterising gravity assist manoeuvres, and another with an inexact a priori satisfaction of those constraints. The global optimisation of trajectories based on the former model will be presented in PART 2 while the global optimisation of trajectories based on the latter model will be presented in PART 3.
- PART 2: THE INCREMENTAL APPROACH (University of Glasgow). This part contains the results of the effort of the group at the Department of Aerospace Engineering of the University of Glasgow. The section contains a general description of the basic idea underneath the incremental approach for the solution

of space trajectory problems. This approach is applied to the solution of problems with an a priori exact satisfaction of the physical GA constraints. Different incremental approaches will be presented. Each one provides a different way of reducing the search space around areas where optimal solutions are most likely to be. In addition, the section presents an incremental approach to the automated planning of multiple gravity assist trajectories where the sequence of swing-by planets and the nature of each transfer leg (low-thrust, ballistic, DSMs) is unknown a priori. A discussion on the algorithmic complexity of the incremental approach is also included.

- PART 3: EXTENDING THE GASP METHOD (University of Reading). This part contains the results of the effort of the group at the University of Reading. The section contains a description of the pruning approach applied to trajectory models with an inexact a priori satisfaction of the GA constraints. A demonstration of the algorithmic complexity of the pruning algorithm is also included.

# References


1. A. V. Labunsky, O. V. Papkov, K. G. Sukhanov, "Multiple gravity assist interplanetary trajectories", *Earth Space Institute Book Series*, Gordon and Breach Science Publishers, 1998
2. N. J. Strange, J. M. Longuski, "Graphical method for gravity-assist trajectory design", *Journal of Spacecraft and Rockets*, vol. 39, n. 1, p. 9-16, 2002
3. A. E. Petropoulos, J. M. Longuski, E. P. Bonfiglio, "Trajectories to Jupiter via gravity assists from Venus, Earth, and Mars", *Journal of Spacecraft and Rockets*, vol. 37, n. 6, p. 776-783, 2000
4. J. A. Sims, A. J. Staugler, J. M. Longuski, "Trajectory options to Pluto via gravity assists from Venus, Mars, and Jupiter", *Journal of Spacecraft and Rockets*, vol. 34, n. 3, p. 347-353, 1997
5. J. W. Hartmann, V. L. Coverstone-Carroll, S. N. Williams, "Optimal interplanetary spacecraft trajectories via a Pareto genetic algorithm", *Journal of the Astronautical Sciences*, vol. 46, n. 3, p. 267-282, 1998
6. P. Rogata, E. Di Sotto, M. Graziano, F. Graziani, "Guess value for interplanetary transfer design through genetic algorithms", in *Proceedings of 13th AAS/AIAA Space Flight Mechanics Meeting*, Ponce, Puerto Rico, 2003
7. A. E. Petropoulos, T. D. Kowalkowski, M. A. Vavrina, D. W. Parcher, P. A. Finlayson, et al., "1st ACT global trajectory optimisation competition: Results found at the Jet Propulsion Laboratory", *Acta Astronautica*, vol. 61, n. 9, p. 806-815, 2007
8. S. M. Pessina, S. Campagnola, M. Vasile, "Preliminary analysis of interplanetary trajectories with aerogravity and gravity assist manoeuvres", in *Proceedings of 54th International Astronautical Congress*, Bremen, Germany, 2003
9. M. Vasile, P. De Pascale, "Preliminary design of multiple gravity-assist trajectories", *Journal of Spacecraft and Rockets*, vol. 43, n. 4, p. 794-805, 2006

10. M. Vasile, R. Biesbroek, L. Summerer, A. Galvez, G. Kminek, "Options for a mission to Pluto and beyond", in *Proceedings of 13th AAS/AIAA Space Flight Mechanics Meeting*, Ponce, Puerto Rico, 2003
11. I. M. Ross, C. N. D'Souza, "Hybrid optimal control framework for mission planning", *Journal of Guidance, Control, and Dynamics*, vol. 28, n. 4, p. 686-697, 2005
12. O. Von Stryk, M. Glocker, "Decomposition of mixed-integer optimal control problems using branch and bound and sparse direct collocation ", in *Proceedings of ADPM 2000 – The 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems*, Dortmund, Germany, 2000
13. D. R. Myatt, V. M. Becerra, S. J. Nasuto, J. M. Bishop, "Advanced global optimisation for mission analysis and design", European Space Agency, Advanced Concepts Team, Ariadna Final Report 03-4101a, 2004
14. D. Izzo, V. M. Becerra, D. R. Myatt, S. J. Nasuto, J. M. Bishop, "Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories", *Journal of Global Optimization*, 2006
15. J. Kennedy, R. Eberhart, "Particle swarm optimization", in *Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia, 1995

# PART 1 MODELLING

## 1.1 Introduction

An engineering design problem can always be tackled with a two-stage approach: problem modelling and problem solution, where often the search for a solution is represented by an optimisation procedure. Modelling is the task of transcribing a physical phenomenon into a mathematical representation.
The modelling stage has a particular influence on the definition and development of preliminary design methodologies since there is always a trade-off between the precision of the required solution and the computational cost associated to its search [1]: different models intrinsically contain different kinds of solutions and can favour or not their identification.
This issue becomes of relevant importance when a large number of good first guess solutions has to be efficiently generated for an exhaustive preliminary assessment of complex engineering problems. In this case, efficiency is quantified as the ratio between number of useful solutions and associated computational time. These considerations apply to trajectory design as well, therefore the two above mentioned stages, which are actually mutually dependent, must be properly defined during the development of an effective design tool for the preliminary investigation of complex interplanetary transfers.
In particular, the modelling process requires the identification of the most important features of the trajectory that will be analyzed and must reproduce the completeness of the problem under investigation, while reducing its complexity. This is a trivial consideration, which has not trivial consequences on the effectiveness of the design phase, since an oversimplified model could lead to a loss of interesting solutions.
On the other hand, a proper mathematical model, which reproduces accurately a physical phenomenon, is likely to require more efficient search methods, in order to find a specific solution. Therefore the problem solution stage needs proper search mechanisms or approaches that allow to locate all relevant solutions in a given solution domain. This raises the additional issue of the completeness of the search: if the problem is at least NP-hard, a complete search could not be practically possible since the number of function evaluations to prove optimality of a solution could grow exponentially with problem dimension.
In the following, the attention will be focused on some simple trajectory models of increasing complexity in order to derive a good compromise between computational cost and solution accuracy. Considering the typical multiple gravity-assist trajectories that have been designed and flown so far, some general features can be considered relevant in order to maintain the required richness of the search space and to identify all the family of solutions that could be potentially interesting for the design of an interplanetary mission. In particular a full 3D model both for the trajectory and for the gravity assist manoeuvres have been developed including deep space manoeuvres (DSM) and using the analytical ephemeris of celestial bodies. The benefits of such a

modelling approach can be seen in the design of missions to Pluto, Mercury or to the Sun, which require to consider the real inclination of the orbit of the planet or of the final heliocentric orbit, and in the design of missions to near earth objects. This particular choice is compared, in terms of search space complexity, to a simpler model in which DSM are neglected. This simple modification prevents from considering some classes of interesting solutions such as, resonant or almost resonant swing-by or free orbits before the encounter with a celestial body.
Since the physics of the Solar system allows to adopt a patched-conic approximation of a multiple gravity assist trajectory, a complete transfer trajectory can be reduced to the sum of a number of smaller sub-problems with a finite number of design variables. As it will be shown in the following, each sub-problem is generally not trivial and may produce complex search domains, typically non-smooth, non-convex and multimodal.
In this section two general trajectory models are presented. Both models describe a multiple gravity assist trajectory with multiple deep space manoeuvres. As will be explained in the remainder of the chapter, the two models allows for different solution approaches. Moreover they could contain a different number and type of solutions: some trajectories that exist in one model might not exist in the other.

## 1.2 Modelling alternatives

The design of multiple gravity assist trajectories with low-thrust and coast arcs requires the definition of a trajectory model and of a search approach. The search approach will use the model to find suitable solutions.
The first step, therefore, is to identify a proper trajectory modelling approach. Fig. 1.1 presents the tree of possible alternative modelling approaches. The trajectory legs could be modelled with a full integration of the dynamic equations or with a sequence of conic arcs and pre-shaped low-thrust arcs liked together. The latter alternative was less computationally expensive and therefore more suitable for global search.
The following choice is to use an unpowered model for the gravity assist manoeuvres or a powered model. In the former case the physical equations defining the outgoing velocity are solved exactly [2], while in the latter case a $\Delta v$ correction at the gravity assist planet is allowed to match the required outgoing velocity with the achievable one [3].
We decided to proceed along both branches and to develop two parallel models, in order to fully understand the implications of each one of the two. The group in Reading followed the powered-GA branch while the group in Glasgow followed the unpowered GA branch.
A further choice for the powered-GA branch was to use a position formulation or a velocity formulation. As will be explained in more details in the following chapter, the velocity formulation suffers from a dependency problem which leads to an exponential growth of the possible alternative paths. For this reason, the position formulation was adopted. Furthermore the position formulation allows for a direct application of the GASP idea [14].
On the side of the unpowered-GA branch the options were to connect the all the sub-regions remaining after the pruning of the solutions space or to analyse each sub-region at the time. As it will be explained in the following chapters the two options have

advantages and disadvantages. In this study we followed the connected branch, though the disconnected one was put on hold for lack of time.
The final decision for both the unpowered-GA and the powered-GA branch was on the search approach.
The alternatives on the powered-GA side were to use either a grid sampling or a multi-start approach. The latter, though stochastic in nature, resulted far more efficient as it is explained in PART 3. The combination of grid sampling for some of the components of the solution vector and multi-start for other components could improve the robustness of the method but was put on hold.
The alternatives on the side of the unpowered-GA side were to look only for feasible regions according to some feasibility criterion or to look directly for all locally optimal solutions. Both branches were explored and will be presented in the following.

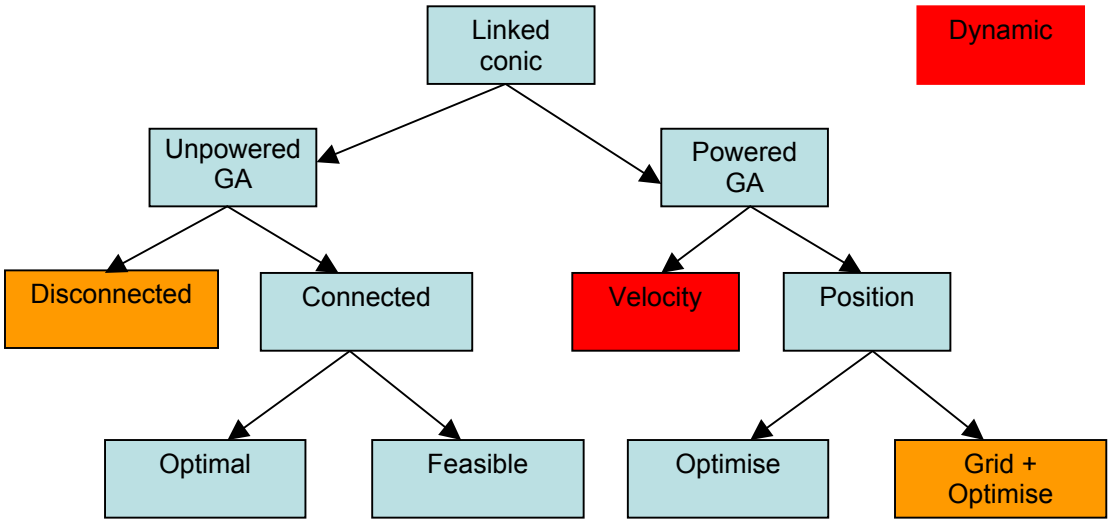



Fig. 1.1. Alternative trajectory modelling approaches: red boxes represent all the discarded methods while orange boxes are methods put on hold which deserve further investigation

## 1.3 Velocity formulations vs. position formulations

The design of a transfer leg with deep space manoeuvre (DSM) can be obtained in different ways. We detailed, in the following, two possible formulations that have radically different implications: velocity formulation and position formulation.

### 1.3.1 Position formulation

In the position formulation, the position vectors of the deep space manoeuvres are assigned (or represent the unknown of the problem in the search for an optimal transfer) and the transfer leg connecting two deep space manoeuvres is computed independently of the other legs either by a Lambert arc or by a shape-based low-thrust arc. The modulus of the velocity discontinuity at the deep space manoeuvre point (or node in the following) is the entity of the $\Delta v$ manoeuvre. Fig. 1.2 is a schematic of the position formulation, the vectors $\mathbf{r}_2$ and $\mathbf{r}_3$ represent the positions of the two deep space

manoeuvres. The modulus and direction of the manoeuvres are computed a posteriori as a result of the discontinuity in the velocity along the trajectory at the nodes.

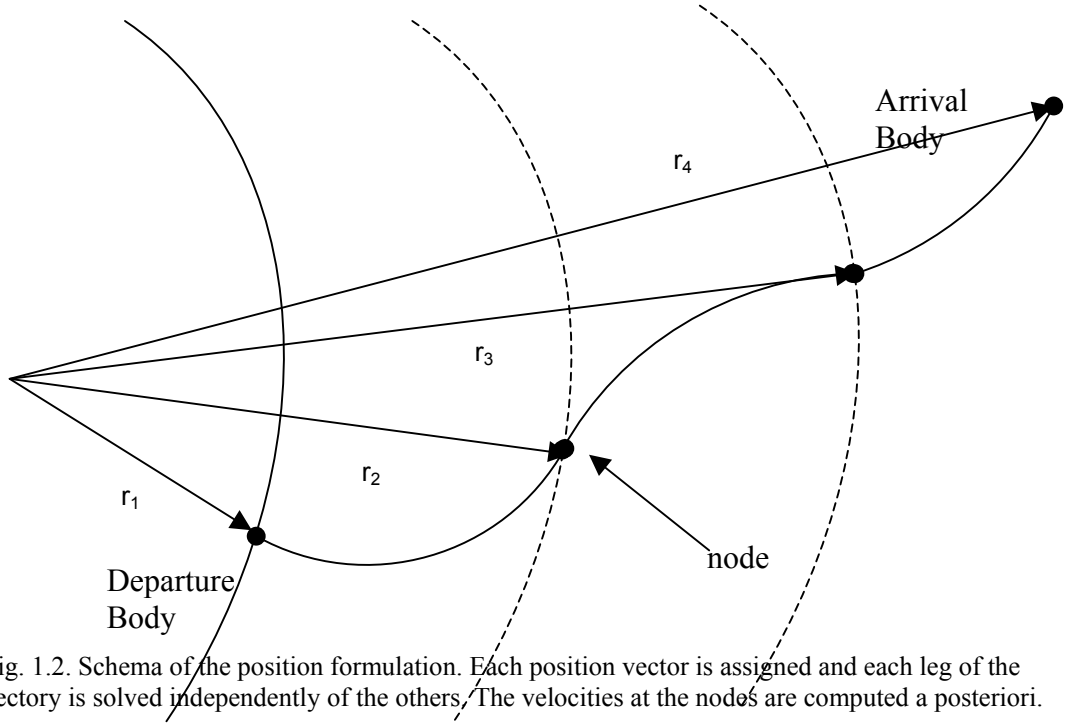



Fig. 1.2. Schema of the position formulation. Each position vector is assigned and each leg of the trajectory is solved independently of the others. The velocities at the nodes are computed a posteriori.

### 1.3.2 Velocity formulation

In the velocity formulation, the modulus and direction of the $\Delta v$ manoeuvres are assigned (or are the unknowns of the problem when searching for an optimal transfer) and the position and velocity at a node is obtained by propagating, forward in time, the state vector (position and velocity) at the previous node. Fig. 1.3 is a schematic of the velocity formulation.

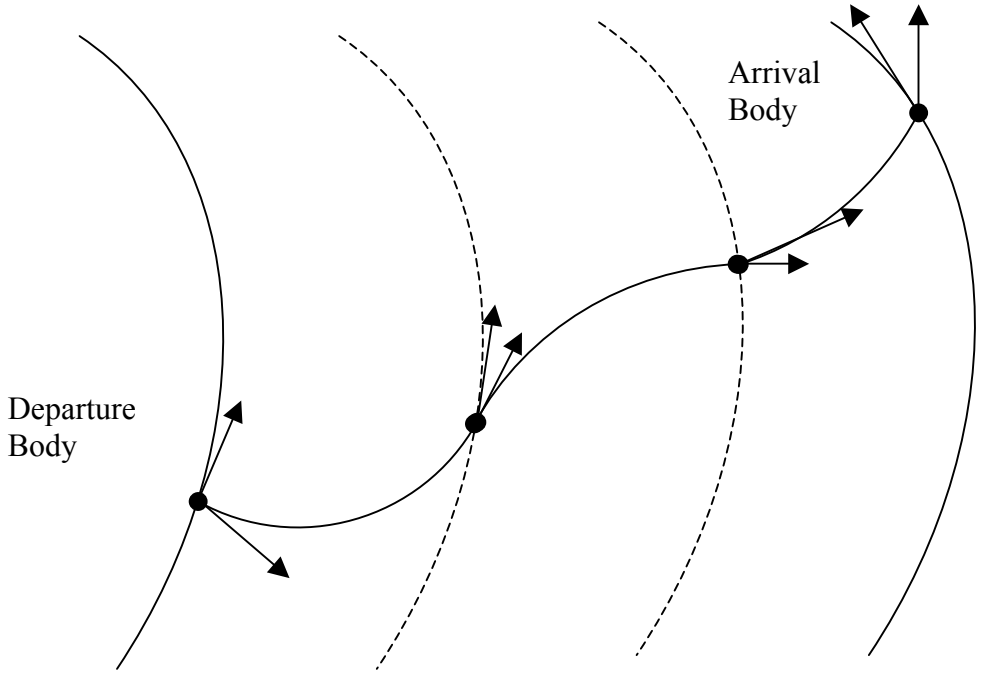



Fig. 1.3. Schema of the velocity formulation. The variation of the velocity at each node is assigned and each leg of the trajectory is obtained by propagation of the state vector. Thus, the position at each node depends on the velocity at the previous node.

### 1.3.3 Implications of the two formulations

The position formulation allows us to compute each arc independently of the other arcs. If we assume that the problem is planar and that the distance of each DSM from the centre of the coordinate system is constant. Therefore, the position of the DSM can be identified by a single angle $\theta_{DSM}$. If we consider three DSMs and we discretise each angle $\theta_{DSM}$ with k steps the number of possible legs from DSM 1 to DSM 2 is equal to $k^2$ and the number of possible legs from DSM 2 to DSM 3 is again $k^2$. The total number of independent legs is, therefore, $2k^2$ and if N deep space manoeuvres are considered the number of possible independent legs would be $Nk^2$. If we see this as a network connecting the departure to the arrival, the number of nodes in the network is Nk+2 and the number of legs connecting the network would be $Nk^2+2k$.

In the case of the velocity formulation, the computation of each arc needs the full state vector at the beginning of the arc. As a consequence if we assume that the modulus of the DSM is given, the problem is planar and the time of flight for each leg is fixed, we can use a single angle $\alpha_{DSM}$ to define the direction of the DSM. Starting from the departure point and discretising the $\alpha_{DSM}$ for each DSM with k steps we get that for three DSMS the number of legs from departure to DSM 1 is k, the number of legs from DSM 1 to DSM 2 is $k^2$ and the number of legs from DSM 2 to DSM 3 is $k^3$. If then we consider N deep space manoeuvres the number of legs is $k^N$.

The implication of the exponential growth of the number of legs for the velocity formulation is that, in the case of a discrete representation of the design variables (in this case the angles $\theta_{DSM}$ and $\alpha_{DSM}$), the generation of the whole solution space would

require an effort that grows exponentially with the number of DSMs. On the other hand the generation of the whole solution space for the position formulation would have a cost that grows polynomially with the number of DSMs.
Furthermore, since each leg of the position formulation can be generated independently of the others, the evaluation of each leg would require only the velocity at the end of the k possible preceding and the velocity at the beginning of the following k legs.
On the other hand, for the velocity formulation, since the position of the DSM depends on all the preceding legs, the evaluation on leg requires the evaluation of all the preceding ones.

## 1.4 Common assumptions

A multi-gravity assist trajectory (MGA) can be defined as a sequence of transfer arcs and swing-bys of gravitational bodies, starting from a departure one and ending at a target one (or a target orbit). Along the transfer arcs, the engine of the spacecraft can be fired to produce a minor change in its velocity vector. Each swing-by, instead, exploits the gravity of the celestial body to produce a major change in the velocity of the spacecraft.
On the scale of the solar system, both the propelled manoeuvres and the gravity-assist manoeuvres can be generally considered instantaneous. Thus, as a first approximation, during each manoeuvre, the heliocentric position of the spacecraft does not change, and coincides with the position of the celestial body at the time of the swing-by, in the case of a gravity assist manoeuvre.
In other words, each manoeuvre has the effect of introducing a discontinuity in the velocity vector, but not in the position vector. The propelled manoeuvres are called deep space manoeuvres (or DSM), and $\Delta v$ is the modulus of the resulting change in velocity.
This particular model of a multi-gravity assist trajectory is called linked-conic approximation since it is made of conic arcs (the transfer arcs) linked together by impulsive changes in the velocity vector (given by the swing-bys).
For each instant of time the position and velocity of the celestial bodies are given by analytical ephemerides, with respect to a heliocentric, ecliptic, inertial reference frame. Therefore, given a sequence of celestial bodies and times of encounter, the position of each gravity assist manoeuvre is fully determined. For the case under examination, all the celestial bodies are planets. At the departure planet, the velocity of the spacecraft is the sum of the launch velocity and the heliocentric velocity of the planet and is normally limited by the launch capabilities.

## 1.5 Trajectory model 1

### 1.5.1 Gravity assist model

As mentioned above, the effect of the gravity of a planet is to instantaneously change the velocity vector of the spacecraft. The relative incoming velocity vector and the outgoing velocity vector, at the planet swing-by, have the same modulus but different directions; therefore the heliocentric outgoing velocity results to be different from the

heliocentric incoming one. In the linked conic model the spacecraft is assumed to follow a hyperbolic trajectory with respect to the swing-by planet. The angular difference between the incoming relative velocity $\tilde{\mathbf{v}}_i$ and the outgoing one $\tilde{\mathbf{v}}_o$ depends on the modulus of the incoming velocity and on the pericentre radius $r_p$ [4]. Both the relative incoming and outgoing velocities belong to the plane of the hyperbola. However, in the linked-conic approximation, the manoeuvre is assumed to occur at the planet, where the planet is a point mass coinciding with its centre of mass. Therefore, given the incoming velocity vector, one angle is required to define the attitude of the plane of the hyperbola $\Pi$. There are different possible choices for the attitude angle $\gamma$; the one proposed in [2] has been adopted (Fig. 1.4): $\gamma$ is the angle between the vector $\mathbf{n}_\Pi$, normal to the hyperbola plane $\Pi$, and the reference vector $\mathbf{n}_r$, that is normal to the plane containing the incoming relative velocity and the velocity of the planet $\mathbf{v}_p$.

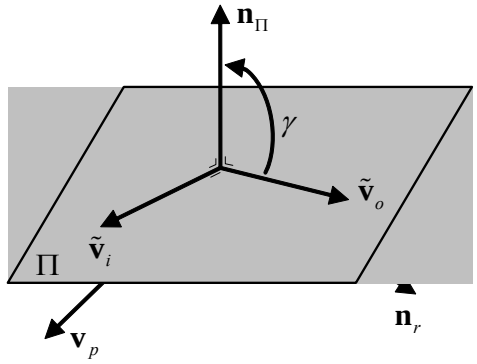



Fig. 1.4: Reference for the swing-by plane angle $\gamma$.

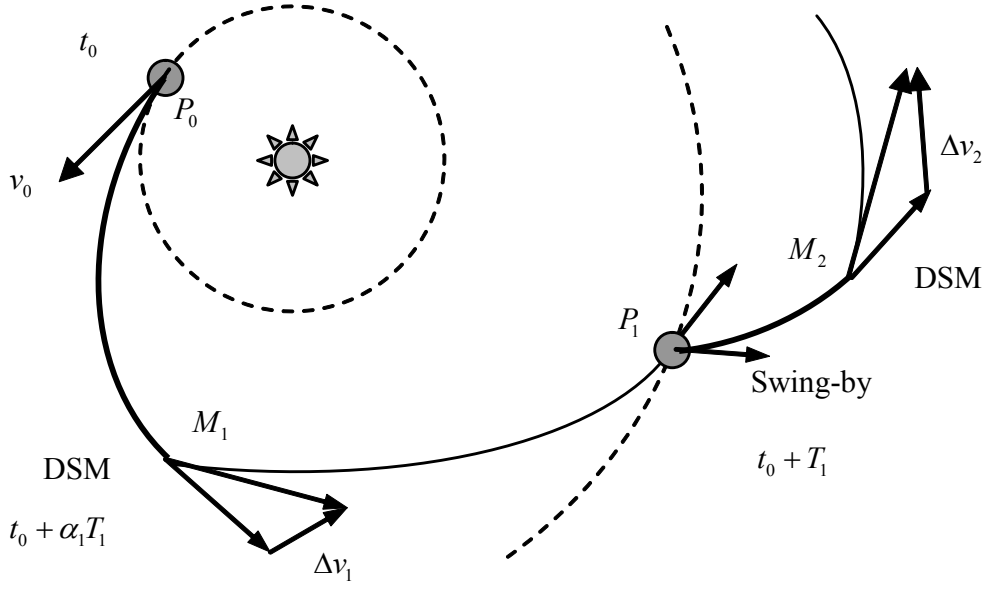



Fig. 1.5: Schematic representation of a multiple gravity assist trajectory.

### 1.5.2 Deep space leg model

A complete MGA trajectory is divided into a number of legs connecting a sequence of celestial bodies (Fig. 1.5). Given a sequence of $N_P$ planets, there exist $i = 1,...,N_P - 1$ legs, each of them beginning and ending with an encounter with a planet. Each leg *i* is made of two conic arcs: the first, propagated analytically forward in time, ends where the second, solution of a Lambert's problem [5], begins. The two arcs have a discontinuity in the absolute heliocentric velocity at their matching point $M_i$. Each DSM is computed as the vector difference between the velocities along the two conic arcs at the matching point. Given the transfer time $T_i$ and the variable $\alpha_i = [0,1]$ relative to each leg *i*, the matching point is at time $t_{DSM,i} = t_{f,i-1} + \alpha_i T_i$, where $t_{f,i-1}$ is the final time of the leg $i-1$. The velocity vector at the departure planet can be a design parameter and is expressed as:

$$\mathbf{v}_0 = v_0 \left[ \sin\bar{\delta}\cos\bar{\theta}, \sin\bar{\delta}\sin\bar{\theta}, \cos\bar{\delta} \right]^T \tag{1.1}$$

with the angles $\bar{\delta}$ and $\bar{\theta}$ respectively representing the declination and the right ascension with respect to a local reference frame with the *x* axis aligned with the velocity vector of the planet, the *z* axis normal to orbital plane of the planet and the *y* axis completing the coordinate frame. This choice allows easily constraining the escape velocity and asymptote direction while adding the possibility of having a deep space manoeuvre in the first arc after the launch. This is often the case when escape velocity must be fixed due to the launcher capability or to the requirement of a resonant swing-by of the Earth (Earth-Earth transfers).

In order to have a uniform distribution of random points on the surface of the sphere defining all the possible launch directions, the following transformation was applied [6]:

$$\begin{aligned} \theta &= \frac{\bar{\theta}}{2\pi} \\ \delta &= \frac{\cos\left(\bar{\delta} + \pi/2\right) + 1}{2} \end{aligned} \tag{1.2}$$

In these equations, $\theta$ and $\delta$ are the free, non-dimensional variables. It results that the sphere surface is uniformly sampled when a uniform distribution of points for $\theta, \delta \in [0,1]$ is chosen.

Once the heliocentric velocity at the beginning of leg *i*, which can be the result of a swing-by manoeuvre or the asymptotic velocity after launch, is computed, the trajectory is analytically propagated until time $t_{DSM,i}$. The second arc of leg *i* is then solved through a Lambert's algorithm, from $M_i$, the Cartesian position of the deep space manoeuvre, to $P_i$, the position of the target planet of phase *i*, for a time of flight $(1-\alpha_i)T_i$. Two subsequent legs are then joined together using the swing-by model. Given the number of legs of the trajectory $N_L = N_P - 1$, the complete solution vector for this model is:

$$\begin{aligned} \mathbf{x} = \big[ & v_0, \theta, \delta, t_0, \alpha_1, T_1, \gamma_1, r_{p,1}, \alpha_2, T_2, ..., \\ & \gamma_i, r_{p,i}, T_{i+1}, \alpha_{i+1}, ..., \gamma_{N_L-1}, r_{p,N_L-1}, \alpha_{N_L}, T_{N_L} \big] \end{aligned} \tag{1.3}$$

where $t_0$ is the departure date.

The formulation of the problem in such way is essential for the pruning approach that will be proposed in the following.

### 1.5.3 Discussion on model complexity

Trajectory model 1 solves explicitly the gravity assist constraints. To do that it requires the incoming velocity before computing the outgoing velocity. In this respect model 1 is equivalent to the velocity formulation for DSMs and presents the same dependency problem. The growth in the number of possible paths is therefore expected to be exponential if a systematic grid decomposition is employed.

The effort of the incremental approach, that will be presented in the second part of this report, is to avoid or reduce the exponential growth of the possible paths.

### 1.5.4 Optimisation formulation

The design of a multi-gravity assist transfer can be transcribed into a general nonlinear programming problem, with simple box constraints, of the form:

$$\min_{\mathbf{x}\in D} f(\mathbf{x}) \tag{1.4}$$

One of the appealing aspects of this formulation is its solvability through a general global search method for box constrained problems.

Depending on the kind of problem under study, the objective function can be defined in different ways. Here we choose to concentrate on the problem of minimising the total $\Delta v$ of the mission, therefore defining:

$$f(\mathbf{x}) = v_0 + \sum_{i=1}^{N_L} \Delta v_i + \Delta v_f \tag{1.5}$$

where $\Delta v_i$ is the velocity change due to the DSM in the *i*-th leg, and $\Delta v_f$ is the manoeuvre needed to inject the spacecraft into the final orbit.

## 1.6 Trajectory model 2

Each MGA trajectory can be decomposed into a number of swing-by phases and deep space flight phases. Each phase can be modelled in many different ways, but under the assumption of patched conics framework, instantaneous swing-bys, and neglecting the spacecraft mass, we can say that for any model, a swing-by phase matches the subsequent deep space flight phase if:

- The swing-by planet is the same as the departure planet of the deep space flight.
- The swing-by time is the same as the initial time of the deep space flight. This also implies that the position of the spacecraft is the same, and matches with the position of the planet at that time (patched conics framework).
- The swing-by outgoing velocity is equal to the initial velocity of the deep space flight phase.

In the same way, a deep space flight phase matches the subsequent swing-by phase if:

- The swing-by planet is the same as the arrival planet of the deep space flight.

- The deep space flight final time is the same of the swing-by time. This also implies that the position of the spacecraft is the same, and matches with the position of the planet at that time (patched conics framework).
- The initial velocity of the deep space flight phase is equal to the swing-by incoming velocity.

Given that the sequence of planets is defined a priori in some way, then the quantities to match are the velocity and the time.
Now, according to model 1, the final velocity in each phase is not a free variable: rather, it is an output of the model, given all the needed parameters. On the other hand, the initial velocity of each phase (both swing-by and deep space flight) is in input of the phase, and must be specified to compute that phase, as pictured schematically in Fig. 1.6.

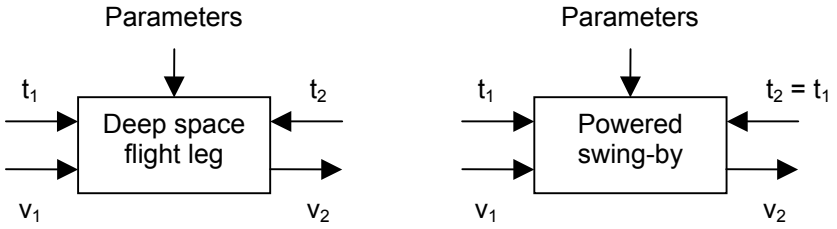



Fig. 1.6. Block representation, with inputs and outputs, of the deep space flight phase and the powered swing-by phase for model 1.

This avoids using this model for decoupling each phase. In fact, to compute any phase the initial velocity is required. It could be arbitrarily set, but then there is no guarantee to find any set of parameters for the preceding phase which give that final velocity. The only way to guarantee the matching with this model is to compute the phases sequentially, such that the final velocity of one phase can be used as initial velocity for the following one.
The alternative is to have a model for which deep space legs and swing-bys can be computed independently, and then matched one another, to create a feasible trajectory. The aim of model 2, in particular, is to remove the dependency, for each leg, on the initial velocity. This is achieved by using a particular model, such that the deep space flight phase only requires the initial and final times, and the parameters, while the initial and final velocities are outputs. On the other hand, initial and final velocities are inputs for the swing-by model, together with the time (Fig. 1.7).

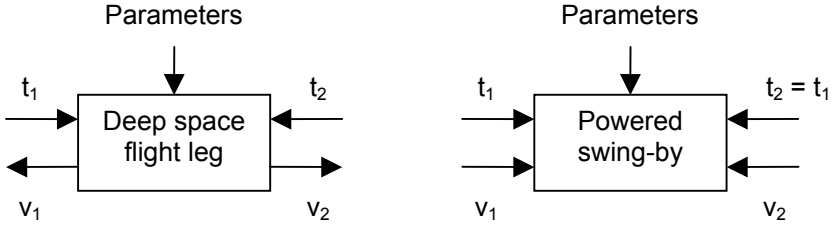



Fig. 1.7. Block representation, with inputs and outputs, of the deep space flight phase and the powered swing-by phase for model 2.

### 1.6.1 Deep space flight with multiple manoeuvres model

This phase is modelled as a sequence of Lambert arcs. The first arc starts at the departure planet, and the last arc ends at the arrival planet. Each arc is connected with the following one in a point in the deep space, where a deep space manoeuvre will occur. The number of deep space manoeuvres in the phase is one fewer than the number of arcs. To compute each Lambert arc, the position (three variables) and the time (one variable) of each DSM shall be specified as parameters of the phase.

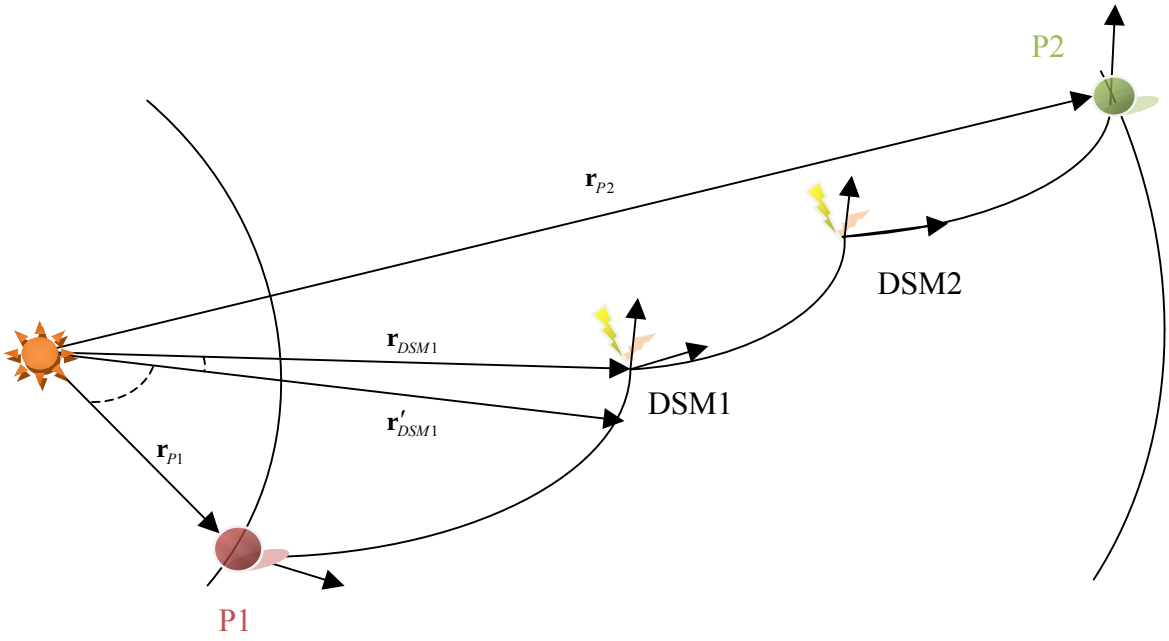



Fig. 1.8. Model 2 requires some parameters to specify the position of each DSM in a deep space flight phase, in addition to the timing.

The position of each deep space manoeuvre is specified in polar coordinates. With reference to Fig. 1.8, the three components are as following:

- Radial distance from the Sun $r_{DSM}$;
- Angle between the position vector of the first planet $\mathbf{r}_{P1}$ and the projection on the plane of the first planet's orbit $\mathbf{r}'_{DSM}$ of the position vector of the DSM $\mathbf{r}_{DSM}$ (*in-plane angle*);
- Angle between the projection on the plane of the first planet's orbit of the position vector of the DSM $\mathbf{r}'_{DSM}$ and the position vector of the DSM itself $\mathbf{r}_{DSM}$ (*out-of-plane angle*).

Rather than inputs, the initial and final velocities of the phase are the outputs of this model. The magnitude of all the deep space manoeuvres is also an output, as this is an important parameter of merit of the considered phase.

### 1.6.2 Powered swing-by model

The swing-by phase of model one shall be able to join incoming and outgoing velocity vectors, which have been computed as final velocity of the preceding deep space flight

phase, and initial velocity of the following one. The idea, then, is to match the two velocity vectors with an hyperbola around the planet A minimum radius shall be given, since there is a limit on the altitude of the spacecraft on the planet (due to the surface of the planet or its atmosphere). The model tries to find a hyperbola, varying the radius of pericentre, such that the velocities at infinity are consistent with the inputs. If the hyperbola is not found, then the solution requires a manoeuvre at the pericentre (thus the name of powered swing-by). The resulting swing-by, in this case, is made by two legs of hyperbolae, connected at the pericentre through a change in velocity (Fig. 1.9).

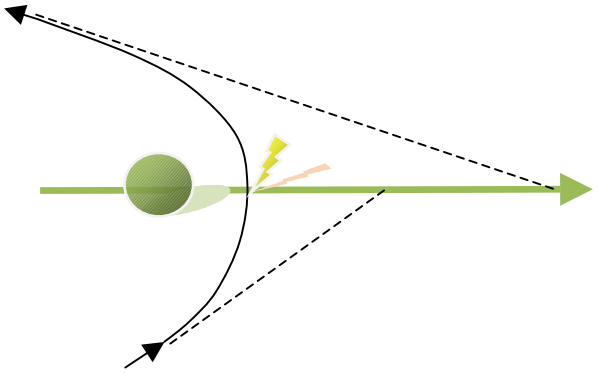


Fig. 1.9. Schematic representation of the powered swing-by. The propelled, instantaneous manoeuvre is preformed at the pericentre of the two hyperbolas.

### 1.6.3 Discussion

Suppose that the problem involves the departure from planet 1, and then, with the help of the swing-by of planet 2, the spacecraft reaches planet 3. In this trajectory there are two deep space flight phases and one swing-by phase.
Given the starting time and ending time for the first deep space flight phase (between planet 1 and 2), and its parameters, the initial and final velocities of the phase can be obtained.
The second deep space flight phase (between planet 2 and 3) can be computed in the same way. Now, if the departure time is chosen to be the same as the arrival time of the preceding phase, then the two phases can be joined with a powered swing-by. This is done easily, since the incoming and outgoing velocity vectors are known: they are the final velocity for the first deep space flight phase and the initial velocity for the second deep space flight phase, respectively.
This model is considered to be “uncoupled”. The reason is that each deep space flight phase can be computed independently from all the others.

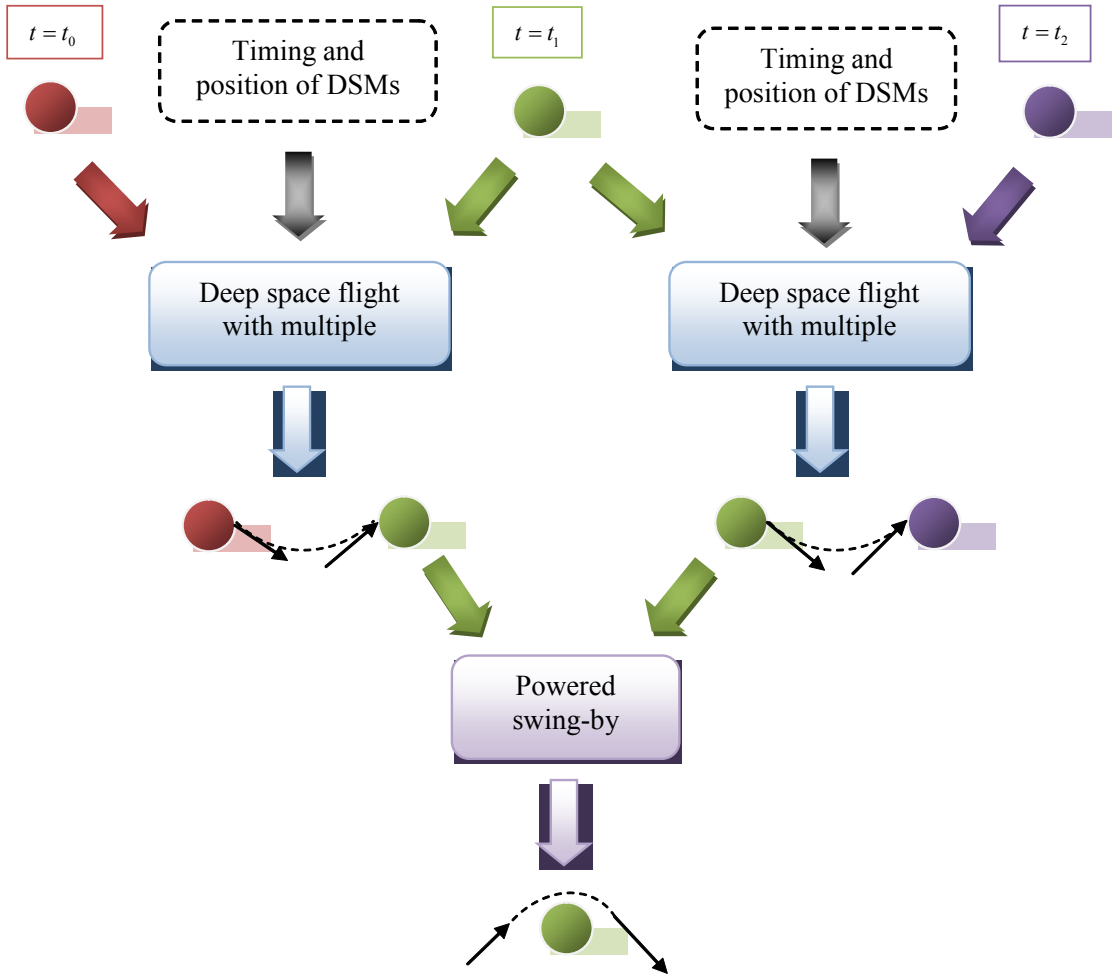



Fig. 1.10. This schema represents the procedure to build a trajectory using model 2. After having fixed the times at the planets, it is possible to compute the deep space flight phases, given the parameters. Then two consecutive deep space flight phases can be joined together through the swing-by.

## 1.7 Block model

In this section, a different model is proposed. Since it allows to decompose a trajectory using basic building blocks, it is a very generic model. Both model 1 and model 2 can be reproduced, together as a combination of them or other different models.

An interplanetary, multiple swing-by, multiple deep space manoeuvre trajectory can be decomposed into phases. In general, these are of 2 kinds:

- Deep space flight;
- Swing-by.

These phases are alternating to compose the whole trajectory. The deep space flight phase can be further split into smaller legs, depending on how the leg is computed (propagation of initial conditions, Lambert arc, …) or depending on the type of arc (coastal arc, low-thrust arc, …). Each phase or leg can be represented by a block, and

different blocks model different parts of the trajectory, or the same part, but using different models. Each block is linked with the following and the preceding one through the spacecraft state (i.e. time, position and velocity).

Let us split the trajectory as a sequence of *blocks*. Each block models, in a certain way, a part of the trajectory. An ordered sequence of blocks defines a complete trajectory, by modelling each part of it in a temporal sequence. All the blocks have duration (they can have zero duration, i.e. be instantaneous): in particular, a characteristic of a block is whether its duration is fixed, or it is a free parameter of the trajectory. Blocks in the sequence cannot be overlapped in time.
Each block has 2 *interfaces*, to join the following block and the previous block (in the temporal sequence). The interface is intended to match the state of the spacecraft between 2 continuous blocks. An interface is characterised by its type: only blocks with the same type of interface can be continuous in the sequence. This means that the interface type is a constraint for matching the blocks in the sequence.
An example of interface for a block which is modelling a deep space flight leg, is composed by position and velocity vectors of the spacecraft. For example, a propagation block with its interfaces is shown in Fig. 1.11.

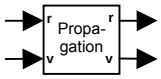



Fig. 1.11. The Propagation block, given an initial position and velocity, propagates forward in time.

An interface type has a set of variables, which must be the same on all the interfaces of that type. Depending on the block, each variable on the interface can be an *input* or an *output* for that block. For the propagation block shown in Fig. 1.11, both position and velocity are inputs on the left hand side interface, while they are output on the right hand side one. This makes sense, as to propagate the trajectory, the initial position and velocity are requested (i.e., before the block), while the result of the propagation is the final position and velocity (i.e. after the block). In this document, an input is shown with an arrow going into the block, while for the output the arrow is pointing outwards. Two blocks can be consecutive in a sequence if they have the same interface type and all the inputs on the interface of one block are outputs on the interface of the other block, and vice versa. When these conditions are satisfied for all the couples of continuous blocks, then the sequence is *feasible*. In the representation of the blocks in the figures of this document, the sequence is feasible if the arrows, representing inputs and outputs for each block, have the same direction on the 2 consecutive interfaces of adjacent blocks, as pictured in Fig. 1.12.
The inputs on both the interfaces must be known to evaluate a block.

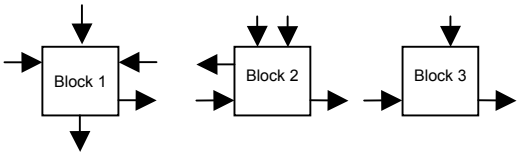



Fig. 1.12. An example of a feasible sequence of blocks with different types of interfaces.

Additionally, a block may have a set of *parameters*, which are also needed to evaluate the block, but they do not belong to any of the interfaces. When a block is evaluated, the outputs are then available on the interfaces. Additionally, a *parameter of merit* can be available as an additional output for the block.
A set of *states* for the spacecraft is defined between each couple of consecutive blocks in the sequence. Differently from the inputs and the outputs, the states shall be known before the evaluation of any block, for all the sequence. A particular state is the time. A block, when evaluated, can read the states at both its interfaces. If the duration of the block is not fixed, then its duration is given by the difference of the state time at its interfaces.
For example, let us consider a block which is computing a Lambert arc (Fig. 1.13). This block is a way of modelling a (part of) a deep space flight phase. Its duration is not fixed, and its interfaces may be defined with 2 quantities: the position of the spacecraft and its velocity. Since to compute a Lambert arc, the initial and final position shall be given, then we can consider that the position on both the interfaces is an input of the block. In the same way, the result of computing the Lambert arc is the velocity vector at its extrema: so, the velocities are outputs of the block. For computing a single revolution, direct Lambert arc, no parameters are needed, other than the initial and final position, and the duration. Thus, this block will not have any additional parameter.

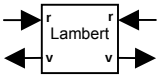



Fig. 1.13. The Lambert block, modelling a Lambert arc. Given the initial and final position (and the time of flight, not included in the interface), computes the initial and final velocity.

For some blocks, some variables on the interface may not be relevant for evaluating the block, and the action of the block has no effect on that variable, i.e. the value of that variable is the same before and after the block. In other words, these quantities are neither input nor output for the block, but they are on the interface. In this case, we say that the block is *transparent* for that variable.
Let us consider for example a block which is modelling an (instantaneous) deep space manoeuvre. The aim of the block is to compute the magnitude of the DSM once the velocity before and after the deep space manoeuvre are known. This means that the velocity is an input on both the interfaces of the block. On the other hand, the block does not explicitly need the spacecraft position, and the position does not change before or after the block. This means that the block is transparent for the position, which is neither input nor output on the interfaces. The variable is represented in the figures as a dashed line connecting the two interfaces (Fig. 1.14).

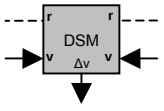



Fig. 1.14. The DSM block, computing a deep space manoeuvre. The block is transparent to variable r.

When a block is transparent with respect to a variable, for example r in the case of block DSM represented in Fig. 1.14, then this variable can be either input or output on one interface of the block, but must be the opposite on the other interface, in order to guarantee the feasibility of the sequence, as shown in Fig. 1.15.

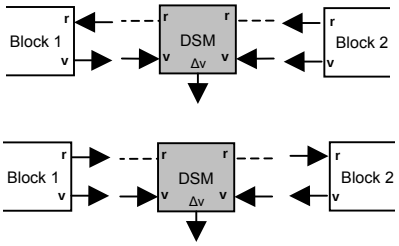



Fig. 1.15. Two possible feasible sequences of blocks.

### 1.7.1 Application to the trajectory

In order to model the entire trajectory using blocks, a set of blocks shall be created, together with their interfaces. The interfaces shall be consistent one another with their inputs and outputs, and allow to reproduce a trajectory.

#### *1.7.1.1 Interfaces*

Three types of interface are used: they are represented in Fig. 1.16.

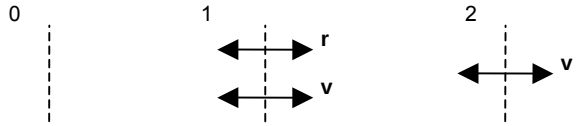



Fig. 1.16. The types of interfaces used to model a trajectory.

Interface 0 has no variables, and it is used on a block which is a terminator of the trajectory or a starter for the trajectory.
Interface 1 is used in the deep space flight, as in this phase the state of the spacecraft is considered to be fully characterised by its position and velocity vectors.
Interface 2 is used when the spacecraft position is supposed to be the same as a planet position. This is the case of a swing-by, for example. Since the planet and the time are states, and thus known a priori, before evaluating any block, the position of the spacecraft can be computed through the ephemerides of that planet at that time. Thus there is no need to include the position vector in the interface.

#### *1.7.1.2 States*

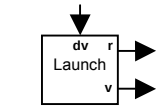
The states, defined between each consecutive block in the temporal sequence, are different than the inputs and outputs, as all the states can be computed before evaluating any block.

Table 1.1. States used to define a trajectory.

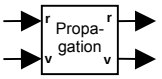
| State | Description |
|---|---|
| Time | Epoch of the interface |
| Previous planet | Planet id of the last encountered planet |
| Following planet | Planet id of the next planet to be encountered |
| Current planet | Planet id, if the spacecraft is considered to be at a planet;<br>0, if the spacecraft is in deep space flight. |

### *1.7.1.3 Blocks*

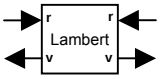
Fig. 1.17 shows the basic blocks composing a trajectory.



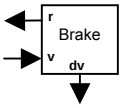

Launch. Fixes position and velocity after the block. It requires the launch dv as an additional parameter.



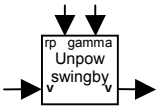

Propagation. Given both position and velocity before the block, it propagates the trajectory for a certain time to obtain position and velocity after the block after the block. It requires the



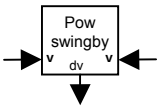

Lambert Arc. It requires the positions before and after the block, and computes the velocities.




Brake. Fixes position (given by the planet ephemeris) before the block, and uses the velocity before to compute the brake manoeuvre. The dv is an additional output.




Unpowered swing-by. Given the incoming velocity, it computes the outgoing velocity. It requires 2 additional parameters to determine the swing-by.




Powered swing-by. It requires both the incoming and the outgoing velocity, and it allows a dv to match the two legs.

Fig. 1.17. Main blocks for modelling a trajectory.

In principle, these blocks are enough for modelling all the phases and legs of the trajectory, according to both model 1 and 2. Even though, this approach highlights that some blocks cannot be consecutive in the trajectory: for example, two Lambert arcs cannot be put next to each other because, even if they have the same type of interface, they both have positions as an input and velocities as an output (Fig. 1.18).

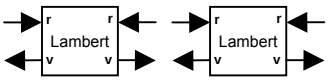
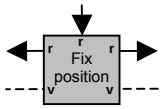
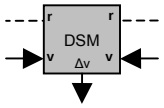



Fig. 1.18. Two Lambert arc blocks cannot match because of the inputs/outputs on their interface.

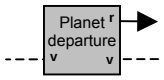
To avoid this problem, some other blocks were introduced.



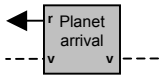

Fix position. Force the spacecraft to pass in a point by fixing the position before and after the block (as given by the external parameter). Transparent to velocity.




Deep space manoeuvre.




Planet departure. Used when leaving a planet, it works as an interface between swing-by blocks and deep space flight blocks. Transparent to velocity, fixes the position after the block.




Planet arrival. Used when arriving at planet, it works as an interface between deep space flight blocks and swing-by blocks. Transparent to velocity, fixes the position before the block.

Fig. 1.19. Additional blocks for modelling a trajectory.

So using these blocks, it is possible to connect 2 Lambert arcs as shown in Fig. 1.20.

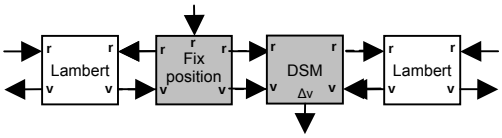



Fig. 1.20. Two Lambert arc blocks connected through a Fix position block and a DSM block.

The input-output configuration on the interfaces of the Lambert arc block forces to add additional blocks to match 2 Lambert arcs. Basically a block which is fixing the position of the spacecraft (Fix position) and a block which is introducing a delta-v manoeuvre (DSM). This configuration makes sense from a physical point of view. In fact, two matching Lambert arcs require the matching point to be given, and in the same point there must be a discontinuity in the velocity vector.

### *1.7.1.4 Feasibility, evaluability and evaluation order*

An ordered sequence of blocks is *feasible* if the interfaces of each couple of continuous blocks are of the same type, and each input parameter on one interface is an output in the other one, and vice versa.
The feasibility of a given sequence does not guarantee that all its blocks can be evaluated. In general we can say that a feasible sequence of blocks is evaluable if it exists an order in which the blocks can be evaluated. This order will be called *evaluation order*, and it is often different than the temporal order of the blocks in the sequence.
The constraint which forbids to evaluate the sequence of blocks in temporal order is the fact that a block needs all the input variables on both interfaces to be known, in order for the block to be evaluated.
Let us consider as an example the interplanetary transfer modelled with the sequence of blocks in Fig. 1.21. This sequence is temporarily ordered, in the sense that the events represented by each block happen in time with the same order of the blocks in the sequence. The sequence is clearly feasible.

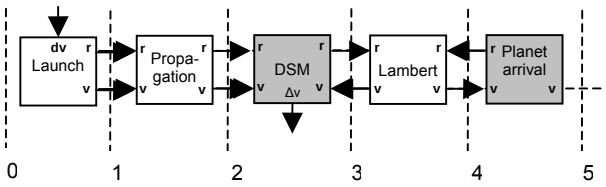



Fig. 1.21. Sections at which the states shall be defined.

Now we can try to evaluate each block of the sequence, considering that the inputs shall be known, and the outputs will be available after the evaluation of each block. The Launch block can be evaluated first, as it has no inputs. Its evaluation makes **r** and **v** available at section 1. **r** and **v** at the same section are inputs for the Propagation block, which can in turn be computed, giving r and v at section 2. The following block, DSM, cannot be evaluated, as the input v at its right hand side (section 3) is unknown. Note that, as the DSM block is transparent with respect to r, the value of this variable is known also in section 3: it is the same as in section 2. The block Lambert cannot be evaluated either, but it is possible to evaluate Planet arrival. This completes Lambert inputs, which in turn completes DSM inputs. Following these criteria, it results that the a possible evaluation order of the sequence is:

Launch, Propagation, Planet arrival, Lambert, DSM.

The evaluation order can be represented graphically as in Fig. 1.22 considering to have an imaginary x axis of the computational time.

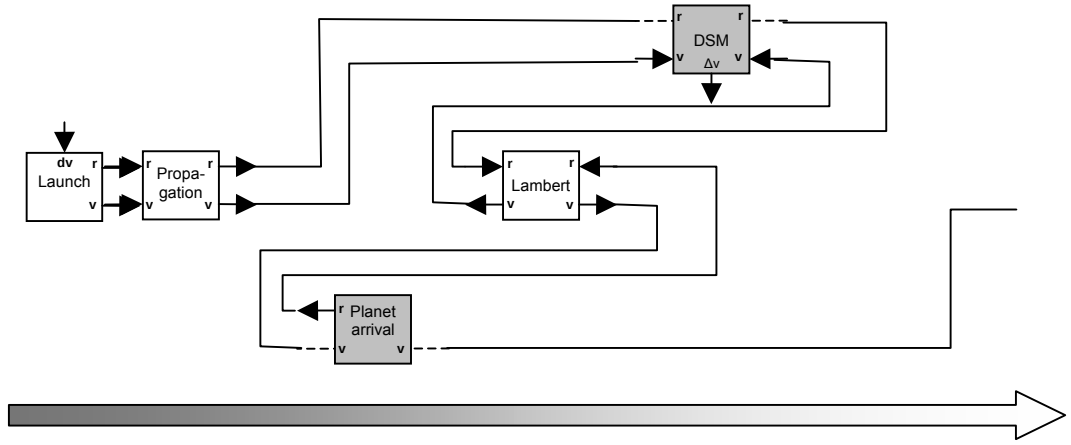



Fig. 1.22. Blocks for the sequence in Fig. 1.21 positioned along a horizontal axis according to their evaluation order.

### 1.7.2 Reproducing other models

#### *1.7.2.1 Model 1*

Model 1 can be reproduced using some of the blocks represented in Fig. 1.17 and Fig. 1.19. The temporal sequence is represented in Fig. 1.23.

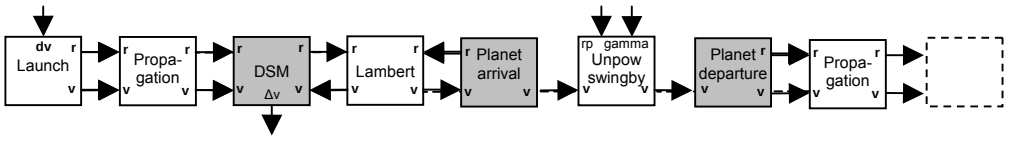



Fig. 1.23. Temporal sequence of blocks reproducing a trajectory according to model 1.

The sequence is evaluable, and it is represented in evaluation order in Fig. 1.24.

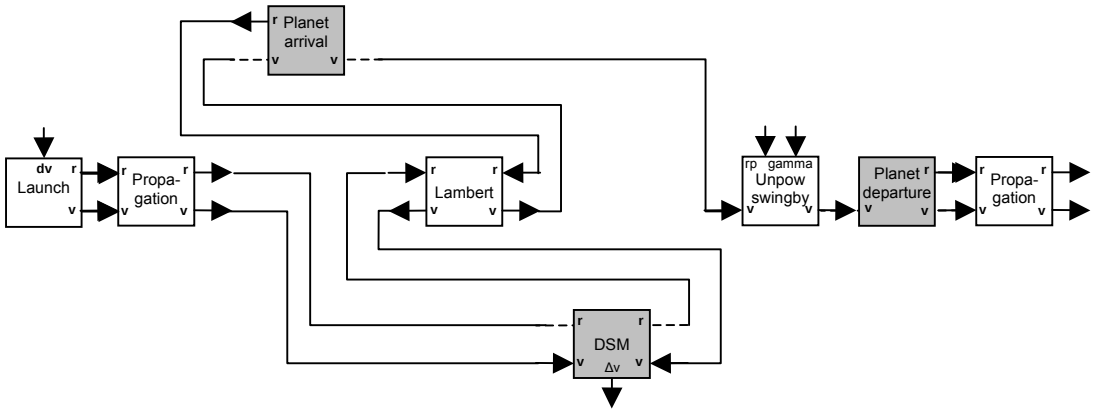



Fig. 1.24. Blocks for the sequence in Fig. 1.23 positioned according to their evaluation order.

### *1.7.2.2 Model 2*

Model 2 can also be reproduced using the block approach. In particular, the deep space flight phase is represented in Fig. 1.25. The number of Lambert arc blocks in the phase is arbitrary.

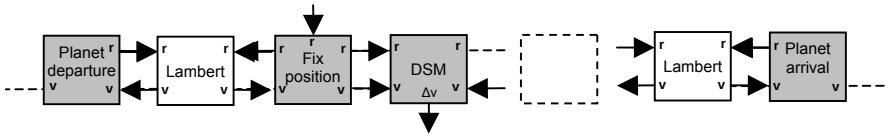



Fig. 1.25. Sequence for a deep space flight phase of model 2

The sequence, regardless the number of Lambert arc blocks, can be evaluated in the order represented in Fig. 1.26.

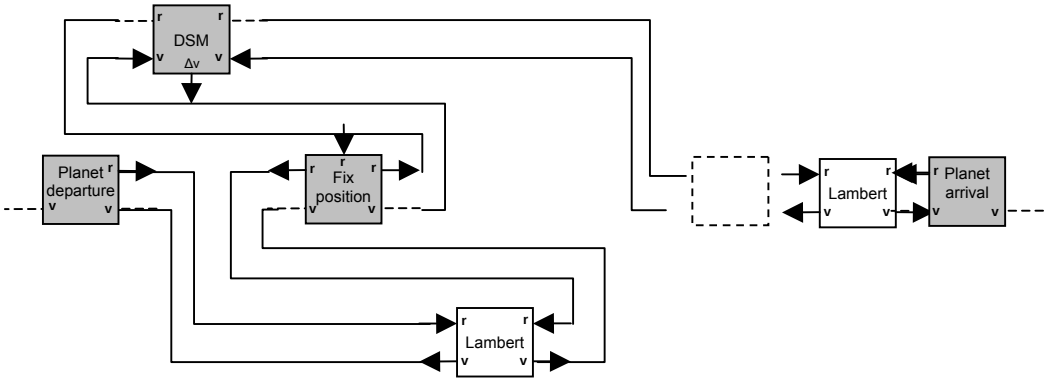



Fig. 1.26. Blocks sorted according to the evaluation order for the deep space flight phase of model 2.

The swing-by phase of model 2 is modelled using the powered swing-by block, which is between the Planet arrival block and the Planet departure blocks (Fig. 1.27). The Powered swing-by block can be evaluated when the following and the preceding deep space phases are computed. This reflects the same approach used in model 2.

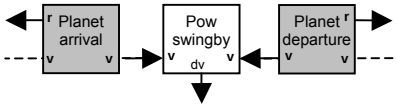



Fig. 1.27. Sequence for a the swing-by phase of model 2

# 1.8 References


1. M. Vasile, "A global approach to optimal space trajectory design", *Advances in the Astronautical Sciences*, vol. 114, n. SUPPL, p. 621-640, 2003
2. M. Vasile, P. De Pascale, "Preliminary design of multiple gravity-assist trajectories", *Journal of Spacecraft and Rockets*, vol. 43, n. 4, p. 794-805, 2006
3. D. Izzo, V. M. Becerra, D. R. Myatt, S. J. Nasuto, J. M. Bishop, "Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories", *Journal of Global Optimization*, 2006
4. A. V. Labunsky, O. V. Papkov, K. G. Sukhanov, "Multiple gravity assist interplanetary trajectories", *Earth Space Institute Book Series*, Gordon and Breach Science Publishers, 1998
5. R. H. Battin, "An introduction to the mathematics and methods of astrodynamics, revised edition", *AIAA Education Series*, AIAA, New York, 1999
6. E. W. Weisstein, "Sphere point picking", Available from: http://mathworld.wolfram.com/SpherePointPicking.html, 2002

# PART 2 THE INCREMENTAL APPROACH

## 2.1 Introduction

This part deals with the incremental approach developed by the team at University of Glasgow. The incremental approach exploits the properties of trajectory model 1. The idea is to prune the solution space incrementally, by alternately adding a leg to the trajectory, and pruning the resulting solution space.

## 2.2 The Incremental algorithm

The generic $\Delta v_i$ in the Eq. (1.5) can be computed once the trajectory is completed up to leg *i*. This means that only the part of the solution vector $\mathbf{x}$ concerning legs 1 to *i* is needed, and the value up to that point is independent of the variables associated to legs $i+1$ to $N_L$. This allows splitting the problem into sub-problems, or *levels*: level 1 is taking into account the $\Delta v$ associated with the launch from the departure planet and the flight to the second planet; each of the following levels takes into account a swing-by and the subsequent leg – including a DSM – to reach the next planet. Let us call $D_{L,i}$ the dimensional slice of the global domain $D$, such that it is composed only by the variables related to level *i*. For the model used here, the corresponding levels, variables and domains are listed in Table 2.1. Let us also define $D_i = \prod_{k=1}^{i} D_{L,k}$, such that the trajectory up to level *i* is defined on the domain $D_i$.

Table 2.1: Levels and related variables.

| Level | Variables | Domain |
|---|---|---|
| 1 | $t_0$, $\theta$, $\delta$, $\alpha_1$, $T_1$ | $D_{L,1}$ |
| 2 | $\gamma_1$, $r_{p,1}$, $\alpha_2$, $T_2$ | $D_{L,2}$ |
| … | … | … |
| *i* | $\gamma_{i-1}$, $r_{p,i-1}$, $\alpha_i$, $T_i$ | $D_{L,i}$ |

Let us introduce a *partial objective function*, for each level, of the form:

$$f_i(\mathbf{y}_i) = f_{i-1}(\mathbf{y}_{i-1}) + \phi_i(\mathbf{y}_i), \quad \mathbf{y}_i \in D_i, \quad i = 1...N_L \tag{2.1}$$

where $\phi_i(\mathbf{y}_i)$ is a function (or local pruning criterion) that is specific to a given level *i* and is used to prune that level. For an MGA trajectory a partial function can simply be defined as follows:

$$f_i(\mathbf{y}_i) = v_0 + \sum_{k=1}^{i} \Delta v_k, \quad i = 1 \ldots N_L - 1$$
$$f_{N_L} \equiv f(\mathbf{x}) \tag{2.2}$$

In this particular case:

$$f_{i-1}(\mathbf{y}_{i-1}) = v_0 + \sum_{k=1}^{i-1} \Delta v_k$$
$$\phi_i(\mathbf{y}_i) = \Delta v_i$$

but in the remainder of this part of the report it will be shown that the definition of a proper function $\phi_i(\mathbf{y}_i)$ plays a very important role in the correct pruning of the solution space.

It is important to stress that the function $f_i$ associated with level *i* depends only on the part of the solution vector related to the legs from 1 to *i*. Moreover, according to Bellman's principle of optimality, if all the trajectory legs from 1 to *i* are optimal, $f_i$ is a lower bound for $f_j$, when $j > i$, and for the whole objective function $f$.

Furthermore we can say that, if $f_i^*$ is the optimal solution of a partial objective function $f_i(\mathbf{y}_i)$, and we define a threshold value $\overline{f}_i > f_i^*$ and a feasible set $\overline{D}_i$ such that:

$$\overline{D}_i = \left\{ \mathbf{y}_i \in D_i : f_i(\mathbf{y}_i) < \overline{f}_i \right\} \tag{2.3}$$

then we can prune out the portion of the solution space that do not belong to $\overline{D}_i$ and consider for level $i+1$ the new solution space:

$$\overline{D}_i \times D_{L,i+1} \tag{2.4}$$

This process is called incremental pruning and $\overline{f}_i$ is called *pruning threshold* for level *i*. What makes this approach interesting is that the evaluation of a partial objective function can be remarkably less expensive than the evaluation of the function *f*, and the associated search space is easier to explore. Thus it is possible to analyse level 1, using $f_1$ on $D_1 \equiv D_{L,1}$, and ideally remove (or *prune*) from the search space all the sets of values for which the partial objective function is above the threshold. The result is a pruned partial domain $\overline{D}_1 \subseteq D_1$. Then the process continues with level 2, considering $f_2$, on $\overline{D}_1 \times D_{L,2}$. Note that this partial domain has a smaller volume than $D_1 \times D_{L,2}$, as there are sets of points in $D_1$ which have already been discarded during the pruning of level 1. The reduction in the search space at level *i* makes the search at level $i+1$ more efficient. At the last level, the complete objective function $f$ is then minimised, on the remaining part of the search domain which was not pruned at previous levels, which is $\overline{D}_{N_L}$.

Different approaches can be used to find the feasible set $\overline{D}_i$ at each level *i*. The proposed incremental algorithm aims, for each level *i*, at the identification of the basin of attraction of the low-laying local minima for each partial objective function $f_i$.

Clearly, care must be taken in defining the different pruning thresholds. While keeping the thresholds too high will result on a light pruning with little improvements on the computational speed of the optimiser, lowering the threshold too much may result in

having the optimal solution left out of the search space. A block diagram of flux of the incremental algorithm is shown in Fig. 2.1.

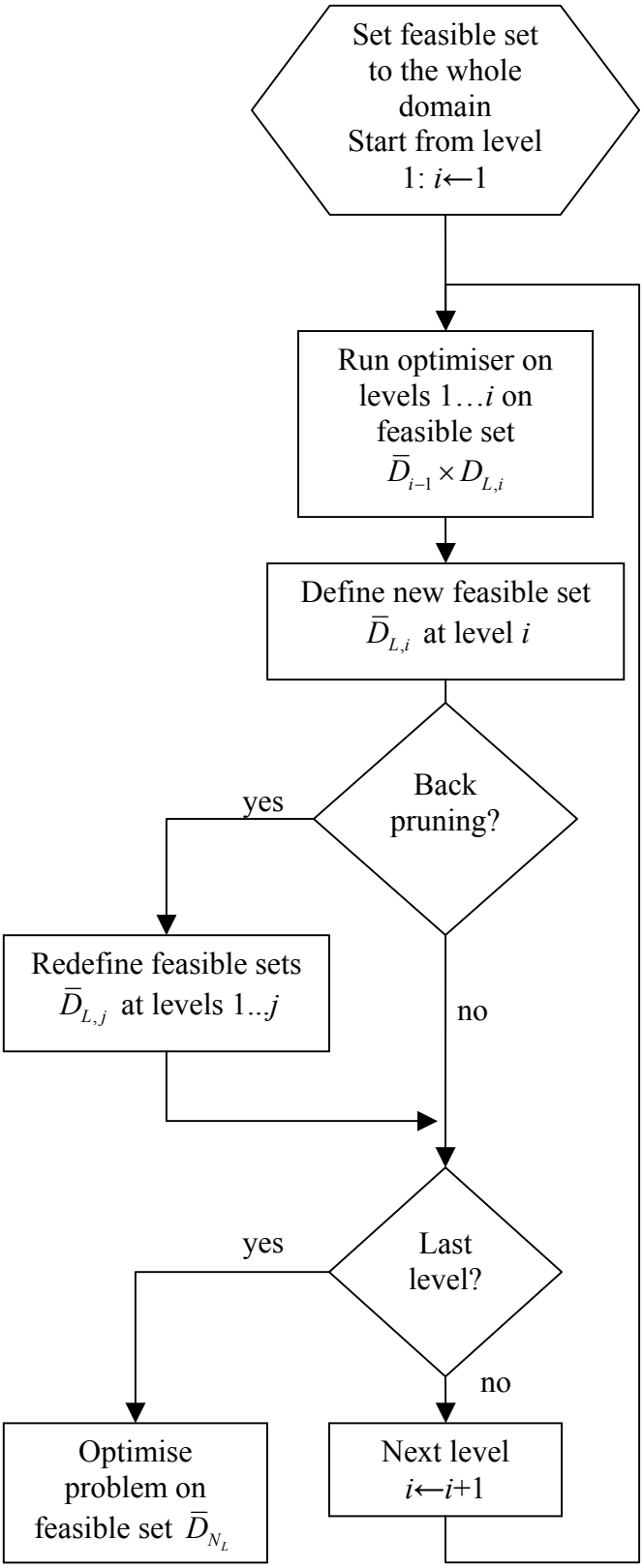



Fig. 2.1. Diagram of flux of the incremental algorithm.

### 2.2.1 Back pruning

In some cases, pruning the search space at level $i$ is not possible since none of the criteria up to level $i$ can be used to discriminate whether a region of the solution space should be pruned or not. In addition, some of the regions in the domain $\bar{D}_i$, which were considered feasible according to the partial objective function $f_i$, then become unfeasible when adding one or more levels. When this occurs, part of the domain $\bar{D}_i$ can be further pruned once the partial objective function $f_j$, $j > i$ is computed at level $j$. Therefore, at each level, a back pruning procedure can used to further reduce the search space of the preceding levels. At level $i$, the algorithm optionally decompose the previous pruned domains into boxes with a smaller edge than the one used before, and some of the boxes are retained while others are discarded depending on the value of the partial objective at level $i$.

## 2.3 Box Collection and Affine Transformation

While the pruning process reduces the search space, the result is that the subsequent optimisation problem is not box-constrained: rather, the search domain becomes the union of all the boxes, on each level. This is clearly a disconnected domain. On the other hand, the optimiser, at the following level, has to be able to search only on those boxes, in order to take advantage of the pruning of the previous levels. Fig. 2.2 shows examples of disconnected and/or overlapped boxes for a 2-dimensional and a 3-dimensional case.

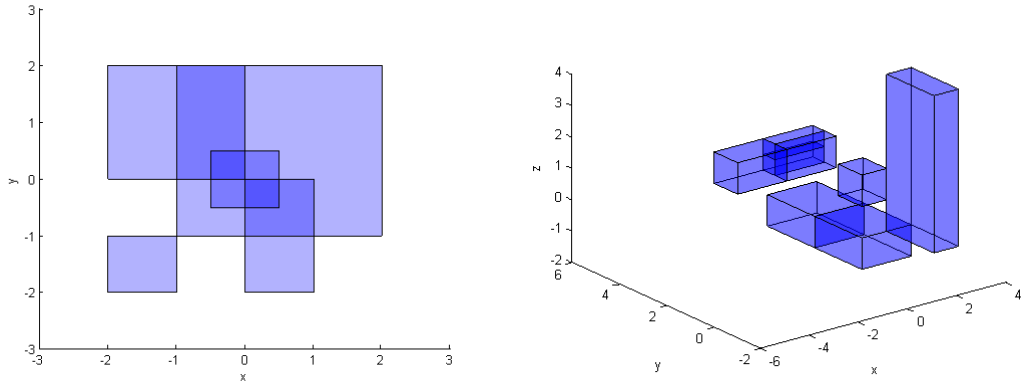



Fig. 2.2. Two examples of domains defined as a set of boxes. The boxes can be partially or completely overlapped.

A possible solution is to collect all the boxes at each level and sample (or search) the collection instead of each single box individually. To this aim, a space transformation was applied, level by level, mapping all the disconnected boxes into a unit hypercube made of connected boxes. If level $i$ is under pruning, then a transformed space is generated for each level $k \leq i$.

The dimensionality of the transformed space is the same as the one of the original level space, so is the number of connected boxes in the unit hypercube. Each box in the real space has a corresponding one in the transformed space, and a linear transformation

allows mapping a point $\bar{\mathbf{x}}$ inside a box in the transformed space into a point $\mathbf{x}$ in the corresponding box in the real space:

$$x_j = \frac{\left(b_{u,j} - b_{l,j}\right)}{\left(\bar{b}_{u,j} - \bar{b}_{l,j}\right)}\left(\bar{x}_j - \bar{b}_{l,j}\right) + b_{u,j} \tag{2.5}$$

for each dimension $j$ of the level under consideration. $\bar{\mathbf{b}}_u, \bar{\mathbf{b}}_l$ are the upper and the lower bounds of the box in the unit hypercube which contains $\bar{\mathbf{x}}$, and $\mathbf{b}_u, \mathbf{b}_l$ are the bounds of the corresponding box in the real space.

Using this transformation of the search space, it is possible to run the search for feasible solutions on the unit hypercube, as schematised in Fig. 2.3. The affine transformation is biunivocal, thus it allows obtaining the point in the real space given a point in the unit hypercube, and evaluating the objective function in that point. In such a way, the optimisation problem becomes box constrained. In addition, only those parts of the search space which have not been pruned out are included in the search.

In general, there exist infinite ways to partition the unit hypercube in a given number of boxes. Each of those may have different properties and drawbacks. In this work, two methods have been studied and used, each one trying to pursue a certain property.

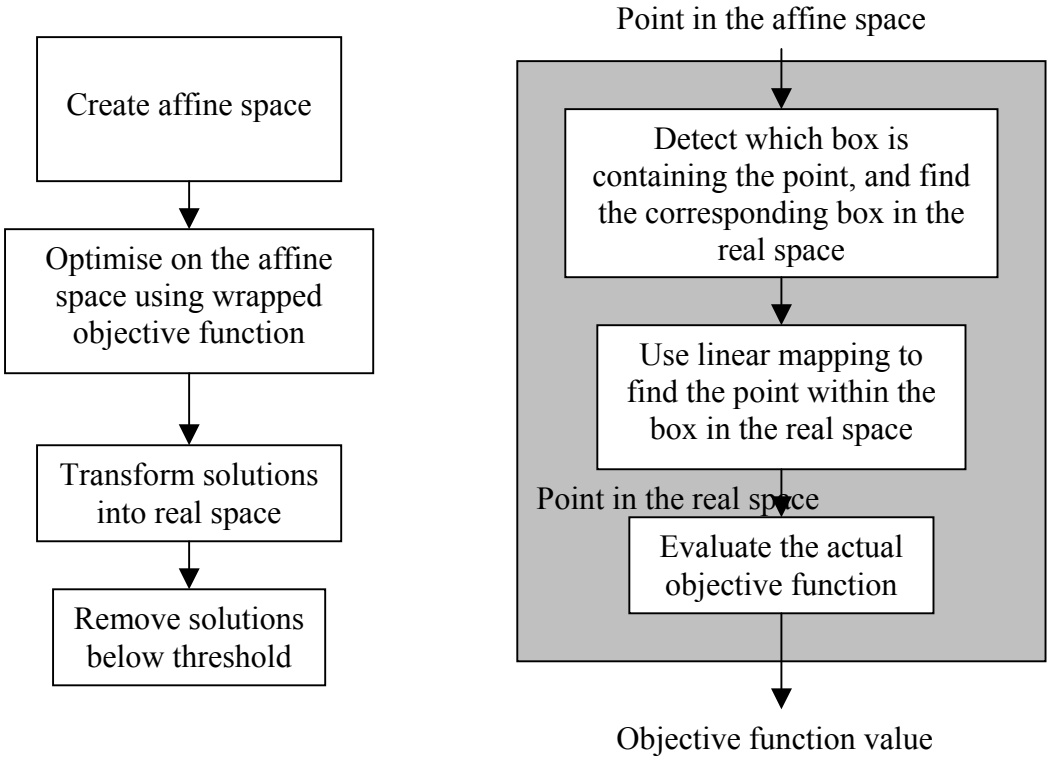



Fig. 2.3. On the left, steps for optimising on the affine space. On the right, the objective function wrapper used to optimise on the affine space.

### 2.3.1 Method 1

The first method aims at partitioning the unit hyper-cube in a way that all the boxes are as similar as possible one another. In general, it is possible to partition a unit hyper-cube into boxes with equal edge length, being $d$ the number of dimensions of the level, and $q$ the number of boxes on that level, only if:

$$t = \sqrt[d]{q}$$

is an integer. $t$ is the number of intervals to consider on each dimension to partition the affine space into $q$ hyper-cubes. In all the other cases, it is always possible to have identical boxes (for example cutting the hypercube along only one coordinate), but the number of subdivisions per coordinate would be uneven.

A method has been developed to try to keep the boxes similar one another in dimensions, and at the same time, with similar edge length:

$$q_L \leftarrow q$$
$$\text{for } j = d \ldots 1$$
$$t_j \leftarrow \text{floor}\left(\sqrt[j]{q_L}\right)$$
$$q_L \leftarrow \frac{q_L}{t_j}$$

This pseudo-code is processing each dimension at a time, and determines $t_j$, which is the number of subdivisions along the $j$-th dimension. $q_L$ is a variable to keep track of the number of remaining boxes to generate, once a dimension has been processed. At the end of the algorithm, if $q_L \neq 0$, then the regular subdivisions could not generate all the required boxes, so further boxes (which will be different in size) are generated by halving existing boxes.

Fig. 2.4 shows the partitioned unit hypercube, for a number of partitions from 1 to 12. The left-hand side of the figure is the case of a 2-dimensional space, while the right-hand side is for a 3-dimensional space. The algorithm works in the same way for an arbitrary number of dimensions.

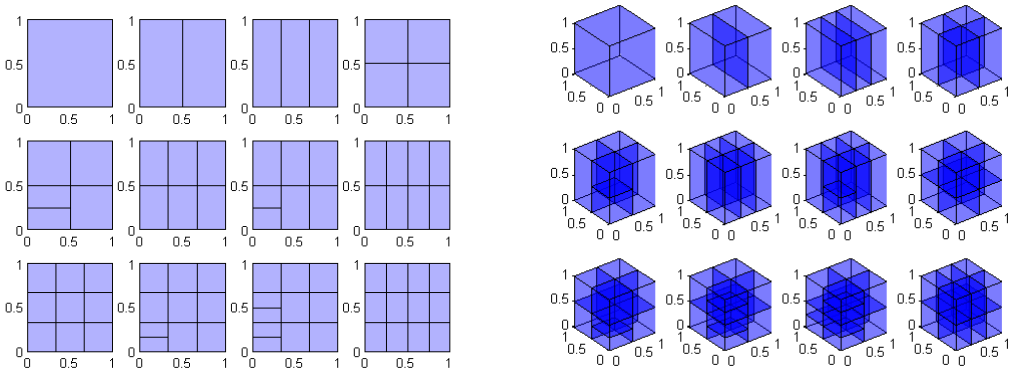


Fig. 2.4. Partitioning of the unit hyper-cube for different numbers of partitions (from 1 to 12), and in the 2D case and in the 3D case, using method 1.

It is noteworthy that, in order to partition the unit hypercube with this algorithm, no information about the size of the boxes in the real space is required.

### 2.3.2 Method 2

The second method for partitioning the unit hypercube aims at preserving the mutual proportions of the volume among the boxes. In this case, in addition to the number of boxes, it is needed to compute the volume of each box in the real space, relative to the total volume of all the boxes. Called the relative volume $V_i$ for a generic box $i$, this must be actual volume of the corresponding box in the unit hypercube, as the total volume of the unit hypercube is unitary by definition. After having sorted the boxes by their volume, in decreasing order, it is possible to start an algorithm. Starting with one box (that is the unit hypercube itself), more boxes are generated by iterative bisection.

The choice of maintaining the volume ratio was made to preserve the sampling probability of the original space $\overline{D}_{L,i}$, when a uniform sampling is performed. Note that this algorithm is partitioning the unit hypercube maintaining the mutual proportions among the volumes, but not among the length of the edges. A 2-dimentional example is shown in Fig. 2.5.

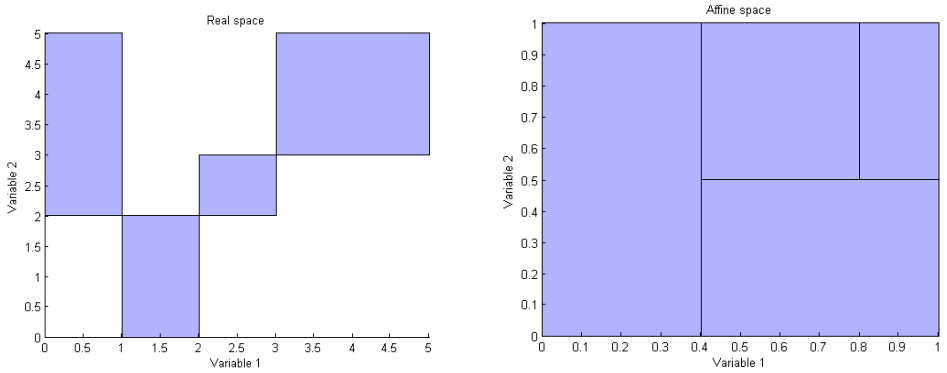



Fig. 2.5. An example of partitioning the unit hyper-cube using method 2. On the left, the real space; on the right, the affine space, partitioned accordingly.

### 2.3.3 Discussion

It should be noted that the objective function seen from the affine space is discontinuous even if it is continuous in the real space (Fig. 2.6). This strongly depends on how the boxes are connected in the affine space. However, if the boxes are disconnected or overlapped in the real space, the objective function in the affine space results to be always discontinuous. In addition the number of local minima in the affine space may be larger than in the real space. Although this might seem a pitfall, it should be noted that in practice a good deal of the minima in the affine space are replica of few minima in the real space. As a consequence there is an increased probability to find a good solution.

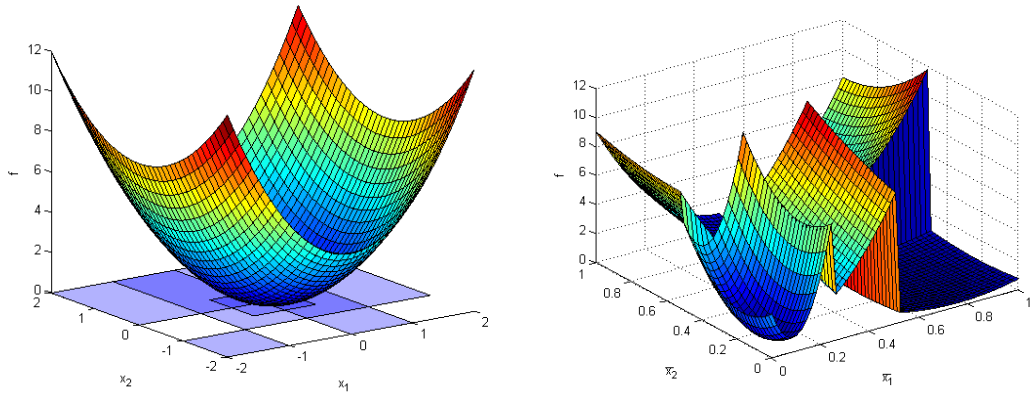



Fig. 2.6. On the left, a paraboloid defined on a set of boxes, and on the right, the corresponding function in the affine space.

## 2.4 Searching for Partial Solutions

As explained above, the pruning process requires the identification of a number of suitable regions of the solution space where optimal solutions are expected to exist. At each level of the incremental approach, these regions can be identified through different methods, in particular we used two conceptually different approaches:

- Low-laying local minima approach. A number of local minima at each level is identified and a region of the solution space is carved around each of the minima.
- Feasible set approach. A number of feasible points, according to a pruning criterion, are identified, clustered and enveloped into a region of the solution space.

Both approaches were applied to the transformed space. For the former approach we used a multi-start algorithm while for the latter we used a global optimiser called EPIC.

### 2.4.1 Multi-start algorithm

This optimiser aims at finding low-laying local minima in the solution space. The search is performed by means of a multi-start algorithm: it generates, for each level of the problem, a uniform random sampling of the search space, and then for each point it starts a local optimisation, using the MATLAB® function *fmincon*.
Once all the points have converged to a local minimum, only those below a given threshold are kept and considered. The threshold must be specified for each level, and it is of course related to the expected value of the partial objective function.

### 2.4.2 EPIC

EPIC is a hybrid algorithm that performs a systematic and automatic search space decomposition into subdomains and explore part of all the subdomains with a stochastic search. The stochastic search is performed by a number of virtual agents implementing a number of individualistic and social behaviours aiming at reaching either a feasible set or a number of local minima. For further details please refer to [24,25].

## 2.5 Pruning Process

The end result of both the approaches used to search for a partial solution, or a feasible set, is a number of points in the solution space. In order to proceed to prune the search space we need to identify a portion of it, containing the points, that is expected to envelope the global optimum.
To this aim we need to defining some bounding boxes containing the points coming from the optimisation process at teach level.

The solution points identified by either one of the optimisers could be low laying local minima or simply solution points below a given threshold (i.e. feasible points). By clustering these points in some way, it is possible to estimate a feasible region, assuming that if a point is feasible, then a region around that point shall be feasible as well.
When pruning level *i*, the solutions are given in the space $D_i$. The feasible sets are defined on each level of the space, so the solution vector is decomposed into levels and a feasible set is created for each level $k$, $k \leq i$. So the function which is defining the feasible set is operating on a space $D_{L,k}$, using the part of solution vectors which are related with level *k*.
Because of how the affine transformation works, the feasible set on each level must be defined as a set of boxes. In theory, a feasible set on a level can have any shape. It is also true that it is possible to approximate any region in an n-dimensional space with a collection of boxes, and a better approximation is obtained with smaller edges of the boxes. Three methods to define a feasible set have been studied through this work.

### 2.5.1 Method 1

For each solution point found by the optimiser, a box is built around the point, such that the point is in the middle, and its edges are parallel to the axes of the variables, and given the size of the edges of the box along each one of the variables of the level.
This method builds a box around each one of the solutions found by the optimiser, without considering whether the solution points are close each other or not. As a consequence, if two solution points are close each other, or even coincident, the two corresponding boxes will be partially or even completely overlapped.
If the solution point is close to the global bounds of the solution space, then the corresponding box is trimmed such not to go outside the bounds. All the boxes, except those near the bounds, have the same size.
Fig. 2.7 represents an example of how this method works, in the case of a level with only 2 dimensions. The red starts represent the solutions found by the optimiser. The blue squares are the 2-dimensional boxes which have been created around the solutions, and their union is the feasible set defined with this method. The boxes have been filled with semi-transparent blue colour, such that a darker colour corresponds to an area in which several boxes are overlapped. From the same figure it is also possible to notice some boxes near the bounds of the search space that have been trimmed, and the thus solution is not in their middle point.

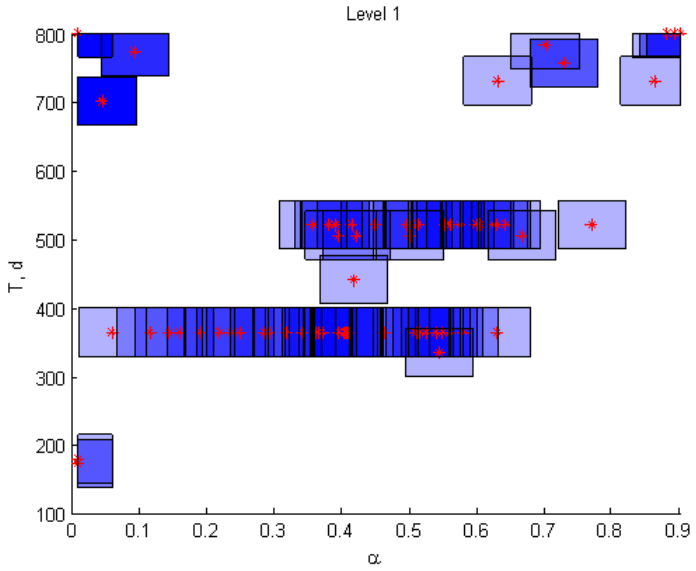



Fig. 2.7. Definition of the boxes identifying the feasible set according to method 1. Red stars represent the optimal solutions, and the blue boxes the feasible set.

The main idea behind this method is that the global minimum of the complete problem should not be too far from one of the low-laying local minima at each level. In particular, the minimum of the global problem must be close enough to the local minimum such that the corresponding box is including it, and thus it is not pruned out from the solution space.

The fact that some boxes are overlapped results in multiple copies of the same part of the solution space into the affine space, once the transformation is done. So the number of local minima in the affine space may be larger than in the real space. Although this might seem a pitfall, it should be noted that in practice a good deal of the minima in the affine space are replica of few minima in the real space. As a consequence there is an increased probability to find a good solution.

A drawback of this method is that the number of boxes is equal to the number of solutions given by the optimiser, and thus can be very high, if an exhaustive search of the solution space has been required. A high number of boxes in each level reflects in a high number of partitions of the corresponding affine space, and so a highly discontinuous function.

### 2.5.2 Method 2

A different approach was followed to develop method 2. In this case, the level domain is ideally subdivided with an orthogonal, uniformly spaced grid. The grid divides the space into boxes. The thickness of the grid, along each dimension of the level, must be specified, and it corresponds to the size of the edge of all the boxes along that dimension.

Once the imaginary grid has been defined, the solution points are introduced, and, for each one of those, the box containing the solution is considered from the ones defined by the ideal grid (Fig. 2.8).

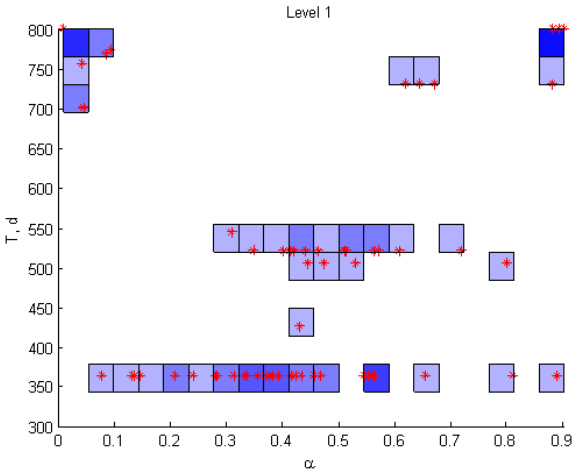



Fig. 2.8. Definition of the boxes identifying the feasible set according to method 1. Red stars represent the optimal solutions, and the blue boxes the feasible set.

With this method, the boxes can either be perfectly overlapped (i.e. coincident), or consecutive, or disjoined. It is not possible to have partially overlapped boxes. All the boxes have also the same size. As in method 1, the number of boxes is equal to the number of solutions. On the other hand, the fact that the boxes are aligned on a grid makes the affine space slightly smoother, especially in the case in which consecutive boxes in the real space are transformed into consecutive boxes in the affine space.

### 2.5.3 Method 3

Method 3 is an evolution of method 2, and tries to fix its flaws.
The boxes are generated in three steps:

- the first step is analogous to what has been done in method 2: the level domain is divided into a number of hyper-rectangles, by means of an ideal grid, given the spacing of the grid along each axis;
- then, all the boxes of the grid which do not contain any solution are discarded, while all the boxes containing at least one solution are saved.
- lastly, all the consecutive boxes are collected together and a wrapping box is built for each one of the collections. This time the box is the smallest box, with edges parallel to the axes, which is wrapping the collection of consecutive boxes.

Fig. 2.9 shows two steps of the box generation with method 3 in a 2-dimensional level domain, while Fig. 2.10 shows the same thing for a 3-dimensional level domain. On the left-hand side of the figures, the red stars are the low-laying solutions identified by the optimiser; the blue boxes are those chosen from the ideal grid, because they contain at least one of the solutions. At this point, all the consecutive boxes are collected together, and the right-hand side of the figures shows the resulting boxes, after the last step. These boxes will be used as the feasible set.

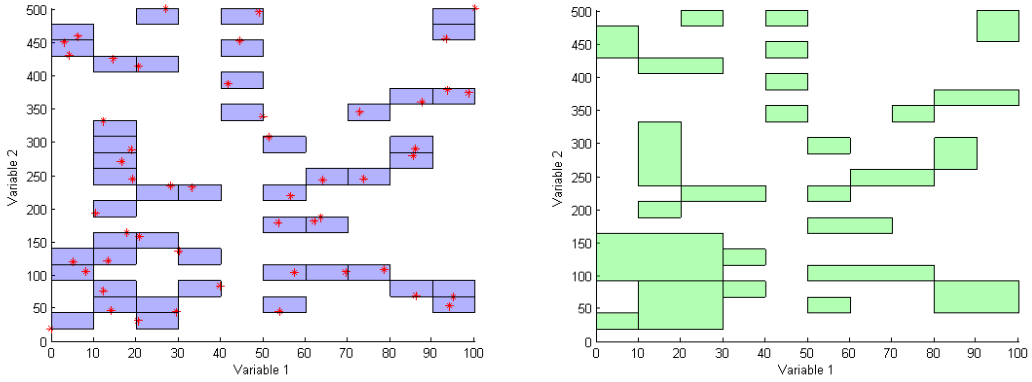



Fig. 2.9. The two steps of generating the feasible set according to method 3. Red stars represent the optimal solutions, the blue boxes are surrounding all the solutions, and finally the green boxes are the feasible set. 2D case.

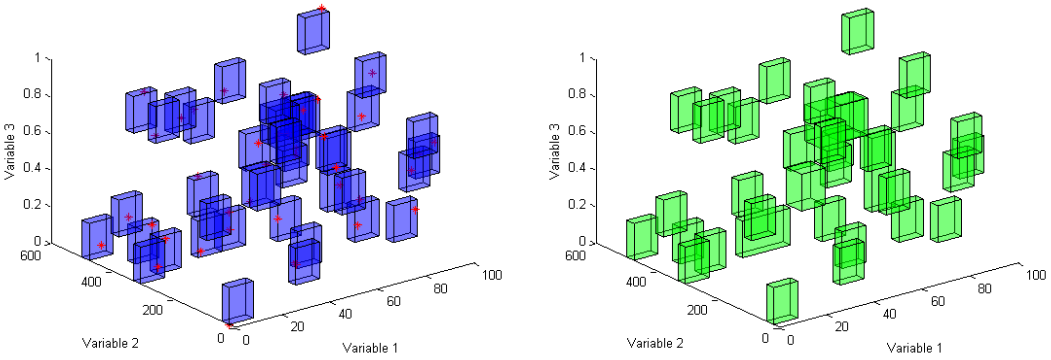



Fig. 2.10. As in Fig. 2.9, for a 3D case.

The aim of the last step is twofold: by collecting several boxes, and creating only one from them, it reduces the number of the boxes; furthermore, it envelopes in one box the regions in which there are many local minima close one another, preserving the local and adjacent structures of the solution space. In fact, the affine transformation maps a box in the real space into another box in the affine space, thus maintaining the shape and the structures of the objective function inside the box. By adopting this method, the boxes don't have the same size, and their number is different than the number of solutions points. More specifically, the number of boxes is most likely to be smaller than the number of solution points, because:

- If two solution points result to be in the same box on the grid, only one box is considered.
- The box collection is likely to further reduce the number of boxes.

## 2.6 Discussion

Before proceeding to the next section, it is worthwhile to examine some of the characteristics of the proposed incremental approach.
One key assumption of the incremental approach is that a complete solution to the MGA problem, i.e. a complete trajectory, can be built by adding individual trajectory legs, starting from departure to the arrival or vice versa. Therefore, although the global

minimum of each sub-problem does not represent the global minimum of the whole problem, we can build the solution space of the whole problem by incrementally adding up the search spaces associated to each sub-problem in such a way that the resulting total search space contains the global minimum.
As will be shown in the following, the objective functions that are used to prune the search space associated to each sub-problem do not directly depend on the chosen objective function for the whole problem. Therefore, the incremental approach is independent of the objective function of the whole problem, but is strongly dependent on the characteristics of the trajectory model.
In particular, for the trajectory model presented in this part of the report, the partial objective function (and pruning criterion) associated to each sub-problem cannot be evaluated without considering all the previous levels. This represents a fundamental difference with respect to what done in [1]. In fact, a trajectory model in which gravity manoeuvres are modelled as powered swing-bys does not need to build the whole solution incrementally (or as a cascade of sub-problems) but each sub-problem can be tackled in parallel with the others. Furthermore, in the proposed incremental approach, the search space is built up incrementally, therefore the number of dimensions of each sub-problem increases as a new level is added to the list. On the other hand, the number of dimensions of each sub-problem in [1] remains constant throughout the whole pruning process.

## 2.7 Incremental Trajectory Planning

This model requires a pre processing to build a problem which can be solved with the incremental approach, with an analogous algorithm as the one discussed in 1.7.
Let us say that we would like an interplanetary transfer between two given planets. By adopting the block model, not only is the choice of planetary swing-bys free, but also the way of modelling each phase.

### 2.7.1 Swing-by sequence definition

A multiple swing-by trajectory, defined using a certain model, is fully characterised given the sequence of planetary swing-bys and a set of other parameters defining the timing and other quantities depending on the model itself. In particular, the choice and the order of planets to swing-by are of great importance, as they change the trajectory completely (and thus strongly affect its characteristics, in particular the delta-v).
It follows that a proper selection of the planetary swing-bys is essential, and shall be done before optimising all the other continuous parameters of the trajectory. Furthermore, tackling the selection of a sequence with a standard optimisation approach is tricky, due to the discrete nature of the variables involved, and to the fact that, once a sequence is chosen, a global optimisation has to be performed in order to assess whether the sequence is promising or not. In addition, the number of different possible sequences of planetary swing-bys for an interplanetary transfer can be very high, which forbids to find an optimal trajectory for each one of them.

A simplified transfer model can be introduced, in order to allow having a very quick assessment of all the possible sequences. If the simplified model is conservative, which means that it gives an optimistic evaluation of each trajectory, it is possible to discard

all the sequences which do not satisfy certain requirements (or are not feasible at all) in a very short time, without the possibility of discarding promising ones. After this step, the complete model can be used to study the remaining sequences.

The departure body and the target body shall be given. The list of possible swing-by sequences is built adding one planet at a time, and then evaluating the feasibility of the sequence using the reduced model. If the sequence is feasible, then the process repeats adding more bodies and checking feasibility, until the target body is reached. At that point, the sequence is considered complete. The algorithm for building the list of possible sequences can be summarised in the following way:

- Initialise the list of feasible sequences A with a single sequence containing only the departure body
- Initialise the list of new sequences B to empty
- Initialise the list of complete and feasible sequences C to empty
- While list A is not empty, do:
    - For each sequence in list A, do:
        - For all the available bodies for swing-by (and the arrival body), do:
            - Add the body to the sequence
            - Check the feasibility
            - If the sequence is feasible, then:
                - If the sequence is incomplete, save it into list B
                - Otherwise, save it into list C
    - Copy list B in list A, then empty list B

The algorithm finishes when there are no more incomplete, feasible sequences. At the end, all the complete, feasible sequences are collected in list C.
Feasibility of each sequence is assessed using a reduced trajectory model, similar to what proposed in [2]. In particular, the following assumptions were adopted:

- Orbits of all the bodies at which swing-by can be performed, and the spacecraft departure body, are considered circular;
- All the orbits and transfers are considered to lie in the same plane (planar system);
- No phasing is taken into account: a swing-by or rendezvous is possible every time the orbit of the spacecraft intercepts the orbit of the body;
- All the swing-bys are performed with the lowest radius of pericentre allowed for the corresponding body, such to achieve the maximum rotation of the relative velocity vector;
- No overturning of the outgoing heliocentric velocity vector after swing-by is allowed. This means that, if the rotation of the incoming relative velocity vector leads to an outgoing relative velocity vector which is on the other semi-plane with respect to the planet velocity, then the rotation which gives the maximum acceleration or deceleration of the spacecraft in the heliocentric reference is considered;
- No other propelled manoeuvres are considered, other than the launch. Swing-bys are responsible for changing the heliocentric energy of the spacecraft.

Given this framework, and a sequence which has to be assessed, the reduced trajectory model is exploited in the following way. The spacecraft is supposed to be at the departure body (so on a circular orbit). The time of launch is not influent, as the body's

orbits are circular and no phasing is considered. The launch is assumed to be tangential (to have the maximum or minimum variation of semi-major axis). The launch is on the same verse of the velocity of the planet if the next planet in the sequence has a bigger radius, otherwise is on the opposite verse. The orbit after launch is subsequently computed: if this orbit intercepts the second body in the sequence, then a swing-by can be performed. Assuming that the planet is in the correct position regardless the arrival time, the incoming relative velocity is computed. Given that the swing-by is performed with the lowest radius of pericentre possible (maximum deviation), but no overturning is allowed, there are two possible outgoing heliocentric velocities: one is higher (positive turning of velocity vector) and the other one is lower (negative turning) than the incoming velocity. Since a swing-by has to be chosen, the same choice proposed by Petropoulos et al. [2] is followed here: if the orbit of the next different planet in the sequence has a bigger semi-major axis, then the positive turning is chosen, otherwise the negative turning is considered. The new orbit after the swing-by is computed and the algorithm continues looking for the intersection with the following body of the sequence. If the swing-bys allow the spacecraft to reach the last planet in the sequence, then we say that the sequence is *energetically feasible*.

This criterion allows to discard some sequences, but the list of possible sequences might still be very big, especially due to some trivial choices: for example, those containing too many resonant swing-bys of the same planet, or those containing swing-bys of farther planets, which, although energetically feasible, can increase the transfer time too much.

So a set of heuristic rules shall be introduced, to make infeasible some sequences, even before the evaluation with the reduced model.

In this work, the following rules were applied:

- The number of resonant swing-bys is limited.
- If the target body's orbit has a bigger radius than the departure's one, then the number of transfer legs going inwards is limited. In the same way, if the target body's orbit has a smaller radius than the departure's one, then the number of transfer legs going outwards is limited.
- If the target body's orbit has a bigger radius than the departure's one, then an inward transfer is possible only between two bodies whose orbits are consecutive in the list sorted by radius. For example, in an Earth-Saturn transfer in the solar system, an Earth-Mercury leg would make the sequence unfeasible, while an Earth-Venus leg is admitted. Equivalently, if the target body's orbit has a smaller radius than the departure's one, then an outward transfer is possible only between two bodies whose orbits are consecutive in the list sorted by radius.
- If the target body's orbit has a bigger radius than the departure's one, then no body with radius bigger than the target one is allowed in the sequence. This is because if it is possible to reach an orbit whose radius is bigger than the target, then it is possible to reach the target, too. Equivalently, if the target body's orbit has a smaller radius than the departure's one, then no body with a radius smaller than target's is allowed.

#### *2.7.1.1 Preliminary results*

The method was tested on a multiple gravity assist transfer from Earth to Saturn. Up to 2 resonant swing-bys were allowed in each sequence and 1 inward transfer was admitted. The resulting list of feasible sequences is given in Table 2.2.

Table 2.2. Feasible sequences for an Earth-Saturn transfer, according to energetic feasibility and heuristic rules.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | E | V | E | E | M | S | | |
| 2 | E | V | E | E | E | M | S | |
| 3 | E | V | E | E | M | M | S | |
| 4 | E | V | E | M | M | M | S | |
| 5 | E | E | V | E | E | M | S | |
| 6 | E | V | E | E | J | S | | |
| 7 | E | V | E | E | E | J | S | |
| 8 | E | V | E | E | J | J | S | |
| 9 | E | E | V | E | E | J | S | |
| 10 | E | V | E | E | M | J | S | |
| 11 | E | V | E | M | M | J | S | |
| 12 | E | V | E | E | E | M | J | S |
| 13 | E | V | E | E | M | M | J | S |
| 14 | E | V | E | E | M | J | J | S |
| 15 | E | V | E | M | M | M | J | S |
| 16 | E | V | E | M | M | J | J | S |
| 17 | E | E | V | E | E | M | J | S |
| 18 | E | E | V | E | M | M | J | S |
| 19 | E | M | E | M | M | M | J | S |

The time needed to compute this list is about 1 second, using MATLAB on a Pentium 3 Ghz computer. It is important to highline that, in the case that only the heuristic rules are applied, there are 249 possible sequences. The energetic feasibility criterion reduces this number to only 19, in a reasonably short time. It is also noticeable that sequence 7 in Table 2.2 is part of the sequence that Jet Propulsion Laboratory selected for the 1st Global Trajectory Optimization Competition, resulting to be the best one. The objective in that case was to reach the asteroid 2001 TW229.

### 2.7.2 Block sequence definition

At this point, we will assume that the planetary sequence is given. The sequence can be the result of a higher-level discrete optimisation, or some heuristics, or previous experience.

We will also assume that a sequence of main blocks (taken from those in Fig. 1.17) generating the trajectory is also given. The sequence shall be consistent with the sequence of planets, that is the number of swing-by blocks (either powered or unpowered) shall be the same of the number of planets to swing-by. The so created sequence of blocks is likely to be unfeasible, like in Fig. 2.11.

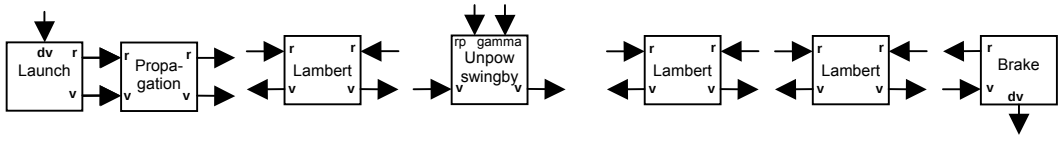



Fig. 2.11. An example of trajectory composed using different blocks, which is unfeasible because of conflicting interfaces between consecutive blocks.

### 2.7.3 Sequence completion

The unfeasible sequence of main blocks is processed and, with the help of some heuristics, additional blocks (those in Fig. 1.19) are added, in order to make the sequence feasible (and evaluable). Table 2.3 shows some of the heuristic rules used to make a given sequence feasible, by inserting additional blocks.

Table 2.3. Some heuristics to make feasible an unfeasible sequence of blocks.

| Unfeasible sequence | Feasible sequence |
|---|---|
| Lambert, Lambert | Lambert, DSM, Fix position, Lambert |
| Propagation, Lambert | Propagation, DSM, Lambert |
| Lambert, Powered swing-by | Lambert, Planet arrival, Powered swing-by |
| Lambert, Unpowered swing-by | Lambert, Planet arrival, Unpowered swing-by |

### 2.7.4 Feasibility and evaluability (evaluation order)

After applying all the available heuristics to make the sequence feasible, the feasibility is checked. It may result that the sequence is still not feasible, because of some errors in the input sequence. For example, if 2 consecutive swing-by blocks (of any kind) are consecutive, that sequence will remain unfeasible, because there is no way to make the sequence feasible by inserting any of the additional blocks from Fig. 1.19. From a physical point of view, this unfeasibility reflects the fact that it makes no sense to have two swing-by phases without any deep space flight in between.
If the trajectory is feasible, then its evaluability is assessed and the evaluation order is computed.

### 2.7.5 Parameters

Once the planetary sequence and the block sequence are given, solution vector for characterising the trajectory is composed by:

- Initial time;
- A time value (duration) for each block which has variable duration.
- All the parameters of all the blocks composing the trajectory;

### 2.7.6 Incremental approach

After computing the evaluation order, the problem can be split into levels, to incrementally prune it.
The concept of creating an incremental approach is the same as explained before. Basically, to prune the search space at a certain level, there is need of a partial objective

function. This will be an indicator, relative to the part of the trajectory which has been computed up to the level under pruning.
Since the trajectory is now composed by blocks, it makes no sense to split the trajectory into levels considering a swing-by and deep space flight phase for each level, as done in 2.2. A rather different approach is followed here, such that an incremental problem can be built regardless the particular sequence of blocks it has been created for describing the trajectory.
The order of computing the blocks is given by the evaluation order. Each time a block is evaluated, all the variables at both its interfaces are determined. In addition, the block can give a parameter of merit. This is used to assess the partial trajectory and to prune the search space. In other words, it is possible to split the problem into levels according to the availability of the parameter of merit in the sequence in evaluation order.
Level *i* is composed by the blocks from 1 to the *i*-th block with a parameter of merit, considering the sequence in the evaluation order. At level *i*, the *i* parameters of merit can be used to create a partial objective function and prune the search space.
Note that this approach is very general, and can be extended to any possible sequence of blocks, regardless if they represent a trajectory or any sequence of actions, under the conditions that the assumptions on the sequence are the same.

The solution vector to characterise the trajectory up to level *i*, is composed by all the variables needed for evaluating all the blocks, from the first to the block with the *i*-th parameter of merit, in the evaluation order sequence. If the blocks are evaluated following the evaluation order, the inputs are available to the block being evaluated. What is needed are then the parameters for the same blocks, and the states before and after each block being evaluated.
The states before and after each block in the level shall be computed. Note that in order to compute the states before and after a block, it might be necessary to define the states at all the sections *preceding* it in the *temporal sequence*. This is particularly true for the time state, in the case of the trajectory.
As an example, let us consider again the fragment of trajectory according to model 1, shown in Fig. 1.21. The same figure shows the sections at which the states are computed. The evaluation order for that sequence is represented in Fig. 2.12, as well as the sections at which the states are defined.

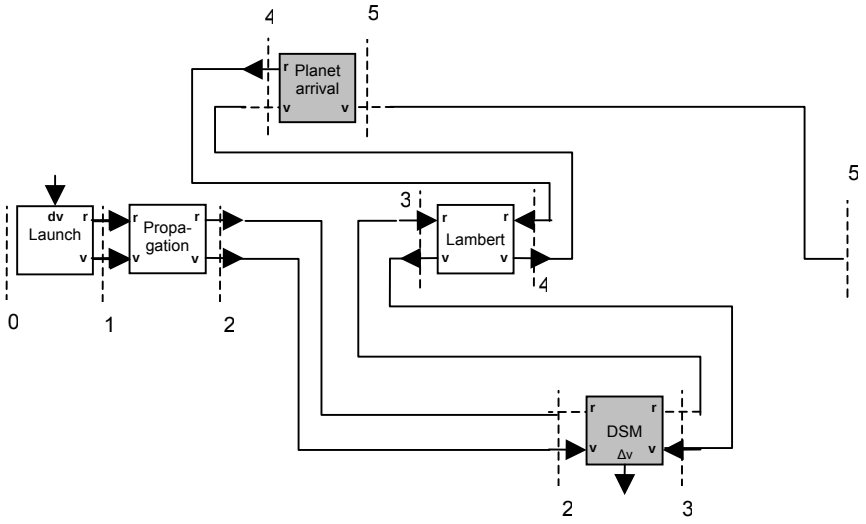



Fig. 2.12. Blocks for sequence in Fig. 1.21 in evaluation order.

To evaluate the Planet arrival block, states at section 4 and 5 are required. These sections, in a temporal sequence, follow the blocks Lambert and DSM, which will be computed afterwards.
If some states are computed from the solution vector (i.e., the time), then the variables of the solution vector needed to compute the states have to be included in the level, even if they are referred to blocks not included in the level.
Summarising, it is possible to state that the solution vector of each level is composed by:

- All the variables needed to compute the states up the last section in the level;
- All the parameters of the blocks in the level.

In the case of the trajectory, this results in:

- Initial time;
- A time value (duration) for each block which has variable duration, and that is in the level and before any block in the level in the temporal sequence;
- The parameters for all the blocks in the level.

In conclusion, for level *i*, given a subset of the solution vector, the set of blocks of the level can be evaluated, following the evaluation order. As a result, *i* parameters of merit are obtained. The parameter of merit of the blocks DSM and powered swing-by is the $\Delta v$, so a partial objective function can be constructed by summing all the parameters of merit. The incremental problem is then fully determined, and can be solved with the tools developed in 1.7.

## 2.8 Results

The incremental approach was tested on a number of problems of increasing complexity with single and multiple gravity assist manoeuvres. In particular, in the following, we report the results obtained for:

- Earth-Venus-Mars (EVM) transfer
- Earth-Earth-Mars (EVM) transfer with two different testing procedures

- Earth-Earth-Venus-Venus-Mercury (EEVVMe) transfer
- Earth-Venus-Venus-Mercury-Mercury (EVVMeMe) transfer

Despite the simplicity of the first two test cases they are representative of two classes of MGA transfers and well illustrate the complexity of these kinds of problems.
The incremental approach was compared to the direct solution of the whole problem (all-at-once approach) with five different global optimization methods, two deterministic and three stochastic. The two deterministic optimisers are DIRECT (Divided Rectangles, [3]) and MCS (Multilevel Coordinate Search, [4]). The stochastic optimisers are DEVEC, an implementation of Differential Evolution [5], PSO, an implementation of Particle Swarm Optimization [6], and a simple multi-start, which takes a suitable number of samples in the search space and optimizes the best one locally by using Matlab® *fmincon*. The optimisers were applied with different settings and with an increasing number of function evaluations. In the following, the optimisers are tested for 20000, 40000 and 80000 function evaluations.

The first three test cases make use of the low-laying minima approach while the last case uses the feasible set approach. For the first three cases, therefore, the incremental approach uses at each level a random sampling of the solution space with Latin Hypercube, and then runs a local optimization from each sample. The local search is performed by using *fmincon*. In both test cases the whole problem is decomposed into two levels. After a set of minima for level 1 is found and a set of boxes is generated, the affine transformation is applied to the subspace at level 1 and the incremental approach proceeds by adding the second level.

### 2.8.1 EVM transfer

The first test case consists of a transfer from the Earth to Mars exploiting a swing-by of Venus. For this test, a simpler version of the trajectory model was used, with no DSM along the Earth-Venus transfer leg. Therefore the problem has dimension 6 and the bounds of the search space are reported in Table 2.4.

Table 2.4. Bounds for the EVM test case

| | LB | UB | Level |
|---|---|---|---|
| $t_0$ [d, MJD2000] | 3650 | 9128.75 (3650 + 15 years) | 1 |
| $T_1$ [d] | 50 | 400 | |
| $\gamma_1$ [rad] | $-\pi$ | $\pi$ | 2 |
| $r_{p,1}$ [planet radii] | 1 | 5 | |
| $\alpha_2$ | 0 | 1 | |
| $T_2$ [d] | 50 | 700 | |

Level 1 computes the first deep space flight phase, while the second adds the swing-by of Venus and the deep space flight to Mars. The objective function $f$ is the total $\Delta v$, which is the sum of the relative velocity at departure and the DSM between Venus and Mars. The problem was initially analyzed by running a multi-start on the whole domain.

A local search was started from a total of 500 starting points, taken with Latin Hypercube, and the 10 best solutions are shown in Table 2.5. The total number of function evaluations needed to compute all the solutions was 494233. The trajectory corresponding to the best solution is shown in Fig. 2.14a.

The three stochastic global optimisers mentioned above were run on the whole problem for 200 consecutive times. In Table 2.6 it has been reported the percentage of times the stochastic optimiser finds a solution proximal to solution 1 in Table 2.5. In addition we report the percentage of times the stochastic optimisers find a solution that is better than the deterministic ones. The key point in the proposed incremental approach is not only to reduce the computational cost but also to increase robustness, i.e. increase the probability to find the global minimum.

The incremental approach starts at level 1 by looking for all local minima for the objective function which is the departure relative velocity at the Earth. A local search was started from a total of 20 random starting points and an equal number of boxes were generated. Fig. 2.13 shows the contour plot of the search space at level 1. The boxes which have been generated by the algorithm are highlighted in dark grey, in semi-transparency. The size of the boxes is arbitrary and was set to a percentage of each dimension.

Table 2.5. The 10 best solutions found with the all-at-once approach for the EVM problem

| Sol. | $\Delta v$ [km/s] | $t_0$ [d, MJD2000] | $T_1$ [d] | $\gamma_1$ [rad] | $r_{p,1}$ [radii] | $\alpha_2$ | $T_2$ [d] |
|---|---|---|---|---|---|---|---|
| 1 | 2.9818 | 4472.013 | 172.2893 | 2.9784 | 1 | 0.5094 | 697.61 |
| 2 | 2.983 | 4473.775 | 170.5335 | 2.9859 | 1.0005 | 0.8611 | 698.1473 |
| 3 | 2.9962 | 4475.217 | 171.1191 | 2.853 | 1.076 | 0.7292 | 692.8782 |
| 4 | 3.0393 | 4480.19 | 167.5824 | 2.8044 | 1.1307 | 0.6371 | 692.5669 |
| 5 | 3.1707 | 4482.079 | 174.6522 | -2.8195 | 1.1885 | 0.4608 | 629.9262 |
| 6 | 3.1708 | 4482.145 | 174.6048 | -2.822 | 1.2033 | 0.4923 | 629.7778 |
| 7 | 3.1719 | 4481.964 | 174.7837 | -2.8076 | 1.106 | 0.6224 | 630.7661 |
| 8 | 3.1884 | 4471.355 | 171.4453 | -3.1416 | 1.019 | 0.5334 | 700 |
| 9 | 3.2217 | 3872.306 | 105.6978 | 2.7087 | 1 | 0.545 | 628.0203 |
| 10 | 3.2536 | 3872.891 | 104.6827 | 2.6838 | 1 | 0.8006 | 627.2178 |

Table 2.6. Solutions and performances of different optimisers on the EVM transfer

| Solver | 20000 evaluations | 40000 evaluations | 80000 evaluations |
|---|---|---|---|
| DIRECT [km/s] | 4.3760 | 4.3730 | 4.3730 |
| MCS [km/s] | 6.7390 | 5.5240 | 5.4080 |
| DEVEC, 200 runs | | | |
| < 3 km/s | 6.5% | 5.0% | 7.0% |
| < DIRECT | 99.5% | 99.5% | 99.5% |
| < MCS | 100.0% | 100.0% | 100.0% |
| Multi-start, 200 runs | | | |
| < 3 km/s | 2.5% | 3.0% | 3.0% |
| < DIRECT | 97.0% | 99.0% | 98.5% |
| < MCS | 100.0% | 100.0% | 100.0% |
| PSO, 200 runs | | | |

| | | | |
|---|---|---|---|
| < 3 km/s | 2.0% | 2.5% | 7.5% |
| < DIRECT | 71.5% | 73.0% | 78.5% |
| < MCS | 100.0% | 96.0% | 93.0% |

The multi-start search of the incremental algorithm was able to identify almost one local minimum for each sinodic period. The number of evaluations to find all the 20 boxes was 516. After applying the affine transformation to level 1 and adding level 2, the whole reduced space was sampled with other 20 random starting points, and a local search was run from each one of them for a total of 8827 function evaluations. The result was that:

- 90% of the 20 best solutions found with the all-at-once approach have the values of level 1 variables included in one of the boxes;
- The best solution found with the incremental approach is the same as the best known solution, i.e. solution 1 in Table 2.5.

In addition, the incremental search has been run twenty consecutive times, obtaining always the same result and the same global minimum. Fig. 2.14 is a representation of the solution for this case.

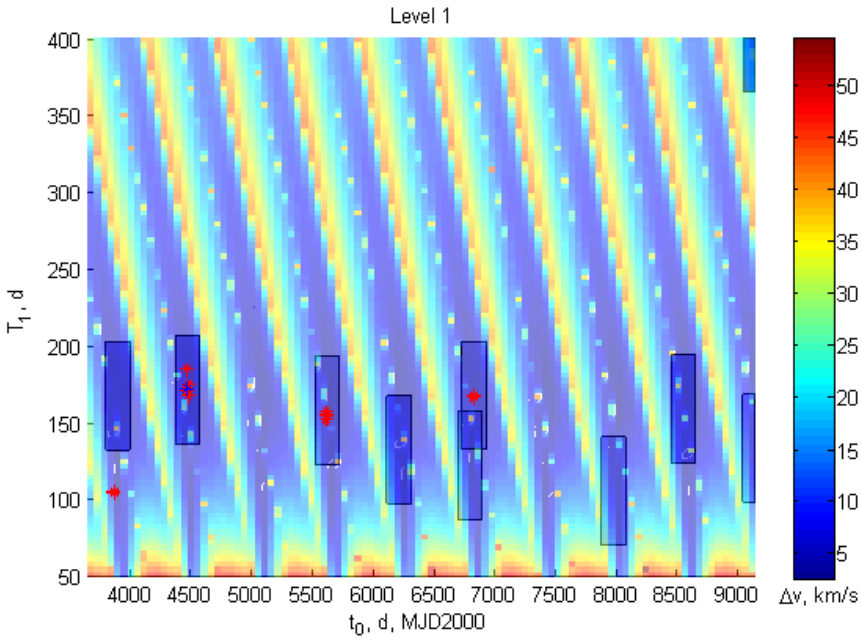



Fig. 2.13. Boxes found after analyzing level 1. The space outside the boxes is pruned. The background gives an idea of the distribution of the local minima

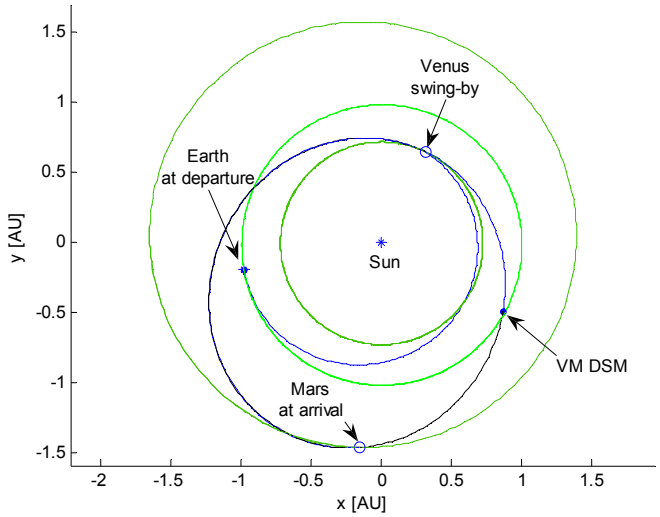



Fig. 2.14. Projection on the ecliptic plane of solution 1 of the EVM test case.

### 2.8.2 EEM Transfer

This single gravity assist test case consists of an Earth-Earth-Mars transfer. Although it is quite simple, the aim of this test is twofold: it demonstrates the effectiveness of the incremental approach, and it is useful to define a particular class of problem-dependent functions $\phi_i(\mathbf{y}_i)$.

The Earth gravity assist is used to increase the kinetic energy of the spacecraft with respect to the Sun when the launch capabilities are limited. In order to gain the required $\Delta v$, the spacecraft has to reach the Earth with a relative velocity vector different from the one at departure. This is achieved with the DSM along the Earth-Earth transfer leg. Thus, the optimal design of the first leg is essential in order to exploit the encounter of the Earth properly, and gain the energy to reach Mars.

The departure velocity vector depends on the launch capabilities, therefore its modulus was set at 2 km/s for this test case, while the non-dimensional declination $\delta$ and right ascension $\theta$ were left free. Being $v_0$ constant, the solution vector has only 5 decision variables on level 1, and $v_0$ can also be removed from all the objective functions without loss of generality. Table 2.7 presents the bounds for the variables of the problem. The global objective function $f$ is the sum of the $\Delta v$ of the two deep space manoeuvres, plus the $\Delta v_f$ needed to inject the spacecraft into an ideal operative orbit around Mars with 3950 km of pericentre radius and 0.98 of eccentricity.

Table 2.7. Bounds for the E-E-M test case.

| | Lower bound | Upper bound | Level |
|---|---|---|---|
| $t_0$ [d, MJD2000] | 3650 | 9128.75<br>(3650 + 15 years) | 1 |
| $\theta$ | 0 | 1 | |
| $\delta$ | 0 | 1 | |
| $\alpha_1$ | 0.01 | 0.99 | |

| | | | |
|---|---|---|---|
| $T_1$ [d] | 50 | 1000 | |
| $\gamma_1$ [rad] | $-\pi$ | $\pi$ | 2 |
| $r_{p,1}$ [planet radii] | 1 | 5 | |
| $\alpha_2$ | 0.01 | 0.99 | |
| $T_2$ [d] | 50 | 1000 | |

In the incremental approach, the whole problem is decomposed into two levels: level 1 consists of the Earth-Earth transfer, while level 2 computes the Earth swing-by and the Earth-Mars transfer leg.

The choice of the partial objective function $f_1$ for the incremental approach at level 1 is tricky. In fact the cheapest way to perform an Earth-Earth transfer is to move from the Earth orbit as little as possible (or not move at all). Therefore, if the sum of the DSM and $v_0$ is chosen as objective to minimise, the optimiser returns solutions with no manoeuvre. These solutions, though, arrive at Earth with a relative velocity that is not suitable to exploit the swing-by properly. Furthermore, it is known from the physics of the problem that the zero-manoeuvre solution is a local minimiser even for the whole EEM transfer. Since the gravity assist manoeuvre requires an accurate timing to reach the swing-by planet with the right incoming conditions, its effect is to narrow down the basin of attraction of each minima. In fact, a gravity assist manoeuvre is more sensitive to a small variation of the variables than a direct transfer. Consequently the gradient of the objective function in a neighbourhood of the local minima is higher and the basin of attraction is expected to be narrower. Now a zero-manoeuvre solution for the E-E leg physically corresponds simply to a delayed departure from Earth, with no gravity assist. All the zero-manoeuvre solutions, therefore, have a much wider basin of attraction. This can be easily verified by applying a general stochastic global optimiser to the whole E-E-M problem. The optimiser will return with a higher probability the zero-manoeuvre solutions if no special condition is imposed on the departure velocity at the Earth.

In order to minimize the $\Delta v$ on the E-M leg, the incoming velocity vector at the Earth should be as such to have an outgoing relative velocity vector aligned with the velocity vector of the Earth (maximum increase in the kinetic energy).

A suitable criterion to optimise the first leg can be found by studying the characteristics of the relative velocity vector at the end of the Earth-Earth transfer. Fig. 2.15 represents the in-plane components (radial $v_r$ and transversal $v_\theta$) of the normalised incoming relative velocity vector for the best solutions found minimising the total E-E-M $\Delta v$ with the all-at-once approach. On the same plot the objective function for the complete problem is also represented.

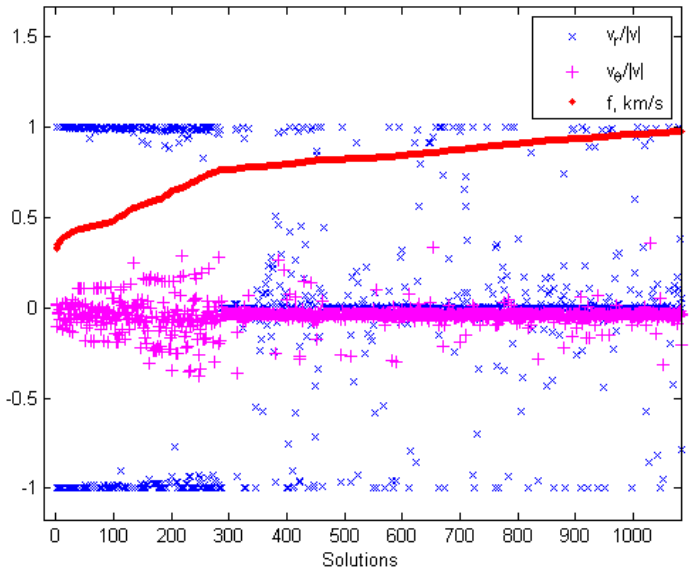



Fig. 2.15. Normalized in-plane components of the incoming relative velocity vector before the Earth swing-by, for the best solutions found, and corresponding objective value.

For the best solutions (from 1 to about 300), the direction of the relative velocity is almost completely radial. Therefore, the following partial objective function was chosen for all the levels in which there is a resonant leg:

$$f_i = \beta \frac{v_\theta^{\,2} + v_h^{\,2}}{v_r^{\,2}} + \sum_{k=1}^{i} \Delta v_k \tag{2.6}$$

This function tries to minimise the DSM while maximising the radial component $v_r$ of the relative velocity before the subsequent swing-by, with respect to the other components $v_\theta, v_h$. $\beta$ was set to 1 km/s. Although this criterion was derived for a specific case, it has general validity and applies to two classes of MGA transfers: aphelion rising gravity manoeuvres and perihelion lowering gravity manoeuvres.
Given the partial objective function in Eq. (2.6), the incremental algorithm was run with 30 randomly distributed starting points for level 1 and 20 for level 2. The threshold for $f_1$ at level 1 has been set to 0.5 km/s. The size of the boxes at level 1 for each variable was set to the values represented in Table 2.8.

Table 2.8. Box size for the E-E-M test case.

| Level under pruning | Box edges at level 1 |
|---|---|
| 1 | 5478.75 d |
| | 0.1429 |
| | 0.1429 |
| | 0.2967 |
| | 95 d |

The length of the edges along all the dimensions is a fraction of the whole search space, except for the edge along direction $t_0$, which spans the entire range. The reason is that the orbit of the Earth is almost circular: therefore a different position along its orbit has

little influence on the arrival conditions at the end of the Earth-Earth leg. Thus, it is not possible to prune along $t_0$ at level 1, i.e. in the E-E leg.

The result of the pruning of level 1 is shown in Fig. 2.16 to Fig. 2.18, which represent the projection of the boxes along variables of level 1. The red stars represent the solutions found by the incremental search engine at level 1. All the search space which is not included in one of the boxes is pruned out, and not considered during the search at the following level.

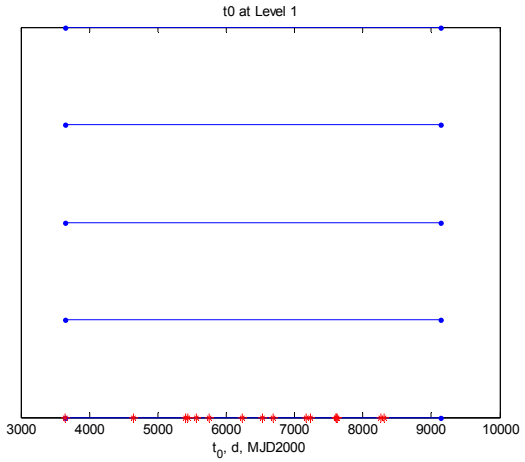



Fig. 2.16: Projection on $t_0$ of boxes and solutions after pruning level 1.

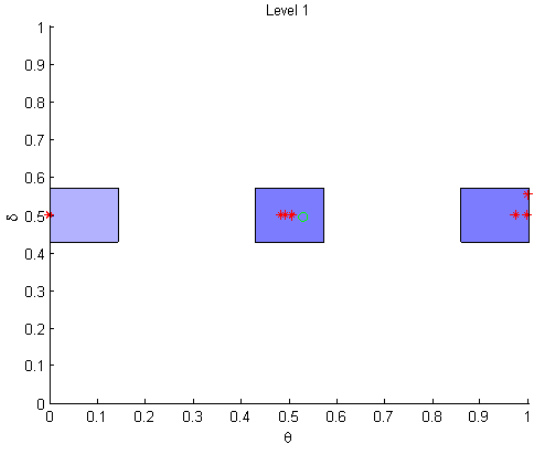



Fig. 2.17: Projection on $\theta, \delta$ of boxes and solutions after pruning level 1.

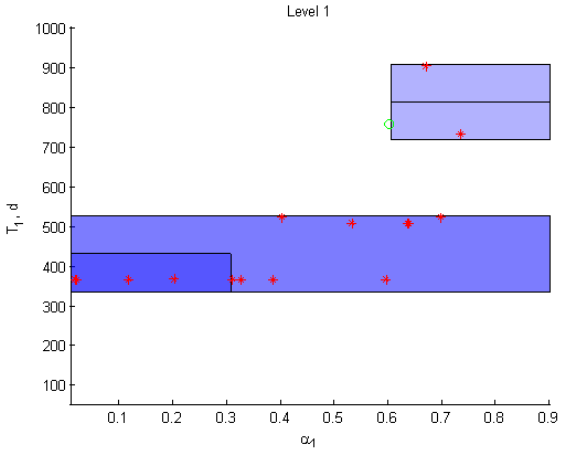



Fig. 2.18: Projection on $\alpha_1, T_1$ of boxes and solutions after pruning level 1.

The fact that $t_0$ is not relevant is clear from Fig. 2.16, as the solutions are spread along the whole time span. Instead, the pruning process reduces the search space along the other variables considerably, in particular $\theta$, $\delta$ and $T_1$. Fig. 2.17 reveals that all the solutions have the non-dimensional declination $\delta$ at 0.5, which means that the launch must be in the Earth orbit plane, and non-dimensional declination $\theta$ equal to 0, 0.5, or 1: these values correspond to a launch excess velocity aligned with Earth orbital velocity (with the same or the opposite direction). Considering the time of flight $T_1$, there are 4 classes of solutions, around 365 d, 510 d, 720 d and 900 d. These local minima have been clustered in 3 boxes, as seen in Fig. 2.18.

The following step of the incremental algorithm is the process of level 2. Since level 2 is the last one in this problem, its pruning is not necessary. All the solutions found by the multi-start are sorted and the best one is considered the best global minimum. The search for the solutions at level 2 takes advantage of the pruning at level 1, and exploits the space transformation.

The smallest $\Delta v$ found by the incremental approach, averaged on the 20 runs, is shown in Table 2.9, together with the same value obtained by running the DE and the multi-start on the complete problem all-at-once. The number of objective function evaluations is also shown, as a parameter of the computational power required to obtain a certain objective value, and thus as an index of the performance of the optimiser. For the incremental approach, the number of function evaluations for each level is shown. The standard deviation of the best-found objective value on the 20 runs is also shown.

The result is that the incremental algorithm finds solutions with a lower $\Delta v$ than DE, PSO and the multi-start, with about 1/10 of the function evaluations.

The trajectory corresponding to the best solution found by the incremental approach is represented in Fig. 2.19.

Table 2.9. E-E-M results for 4 different approaches, values computed on 20 runs.

| | Average no. of function evaluations | Best $\Delta v$ [km/s] | |
|---|---|---|---|
| | | Average | Standard deviation |
| DE | 200070 | 1.591 | 0.136 |

| | | | |
|---|---|---|---|
| all-at-once | | | |
| PSO all-at-once | 200000 | 1.556 | 0.238 |
| multi-start all-at-once | 210217 | 1.268 | 0.137 |
| Incremental | 6097, 8519 | 1.171 | 0.081 |

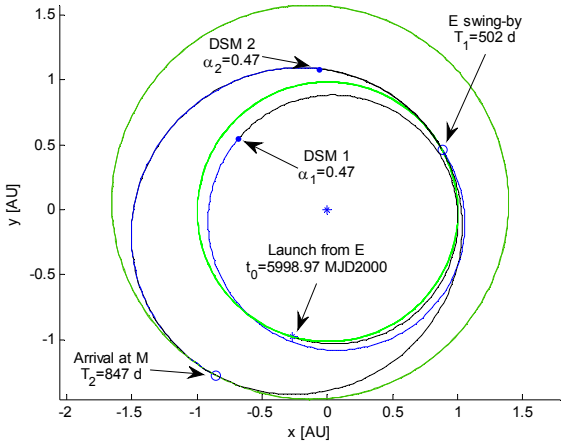



Fig. 2.19. Projection on the ecliptic plane of the best solution found by the incremental algorithm. The total $\Delta v$ is 1.08 km/s.

### 2.8.3 EEVVMe Transfer

This sequence to reach Mercury includes three swing-bys, 2 of which are resonant. It is the same sequence chosen for the MESSENGER mission [7], and is likely to be chosen for the BepiColombo mission [8], although the latter will use low-thrust propulsion.
The launch excess velocity was fixed to 1.5 km/s, and no orbit insertion manoeuvre was considered at Mercury, because other resonant swing-bys may be added to further slow down the spacecraft. The objective function is then the sum of the deep space manoeuvres in each leg. The bounds for this problem are shown in Table 2.10. The launch window and the time of flights intervals were chosen to include the BepiColombo baseline mission option [16].

Table 2.10: Bounds for the E-E-V-V-Me test case.

| | Lower bound | Upper bound | Level |
|---|---|---|---|
| $t_0$ [d, MJD2000] | 4500 | 5500 | |
| $\theta$ | 0 | 1 | |
| $\delta$ | 0 | 1 | 1 |
| $\alpha_1$ | 0.2 | 0.9 | |
| $T_1$ [d] | 350 | 600 | |
| $\gamma_1$ [rad] | $-\pi$ | $\pi$ | 2 |

| | | | |
|---|---|---|---|
| $r_{p,1}$ [planet radii] | 1 | 5 | |
| $\alpha_2$ | 0.01 | 0.99 | |
| $T_2$ [d] | 300 | 450 | |
| $\gamma_2$ [rad] | $-\pi$ | $\pi$ | |
| $r_{p,2}$ [planet radii] | 1 | 5 | 3 |
| $\alpha_3$ | 0.01 | 0.99 | |
| $T_3$ [d] | 150 | 300 | |
| $\gamma_3$ [rad] | $-\pi$ | $\pi$ | |
| $r_{p,3}$ [planet radii] | 1 | 5 | 4 |
| $\alpha_4$ | 0.595 | 0.733 | |
| $T_4$ [d] | 750 | 850 | |

The 4th leg was required to perform 6 complete revolutions around the Sun. To this aim, the bounds on $\alpha_4$ were restricted such that the propagated part of the leg can perform at least 3 complete revolutions, while the subsequent Lambert problem is solved searching for a 2-complete-revolution solution.
For the incremental approach, 100, 100, 100, 200 starting points for levels 1 to 4 respectively were used. The size of the boxes for level 1 was set to a fraction of the span of the space (apart from $t_0$), as shown in Table 2.11. At the pruning of level 2, the back pruning was used, and the boxes were re-generated also on level 1, with their edge on variable $t_0$ reduced to 1/10. The reason is that the E-V leg introduces some constraints on the phasing of the Earth-Venus system, and this reduces dramatically the range of the possible launch dates, in order to have a low $\Delta v$. For the variables of the levels 2 to 4, the size of the boxes was kept fixed, as in Table 2.11.

Table 2.11: Box size for the E-E-V-V-Me test case.

| Level under pruning | Box edges at level 1 | Box edges at level 2 | Box edges at level 3 | Box edges at level 4 |
|---|---|---|---|---|
| 1 | 1000 d<br>0.2<br>0.2<br>0.2333<br>50 d | | | |
| 2…4 | 100 d<br>0.2<br>0.2<br>0.2333<br>50 d | 1.25 rad<br>1.33<br>0.29<br>30 d | 1.25 rad<br>1.33<br>0.29<br>30 d | 1.25 rad<br>1.33<br>0.046<br>20 d |

The objective function in Eq. (2.6) was chosen for searching the solutions on level 1 and 3 of the incremental approach. These levels correspond to the resonant-swing-by legs E-E and V-V respectively. For levels 2 and 4, the sum of the $\Delta v$ was chosen. Local minima above 1, 1.1, 1.2 km/s were discarded at levels 1, 2, 3 respectively.

As in the E-E-M case, the pruning of level 1 does not identify any particular launch window, even if some periodicity is visible due to the eccentricity of the Earth orbit (Fig. 2.20). As seen in Fig. 2.21, the incremental algorithm clearly identifies an in-plane, tangential launch. Fig. 2.24 shows three classes of solutions with 3 possible time of flights $T_1$, clustered into 2 sets of boxes. The solutions are spread in a wide range on $\alpha_1$.

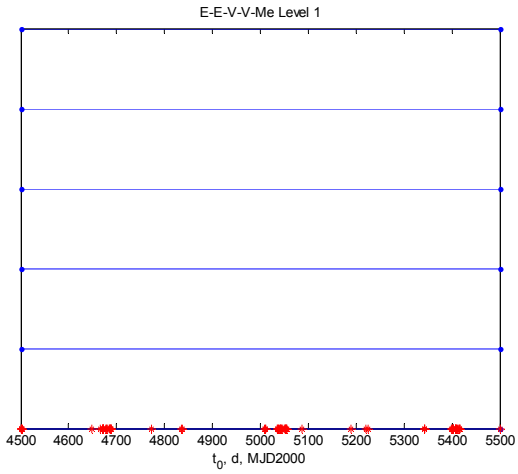



Fig. 2.20: Projection on $t_0$ of boxes and solutions after pruning level 1.

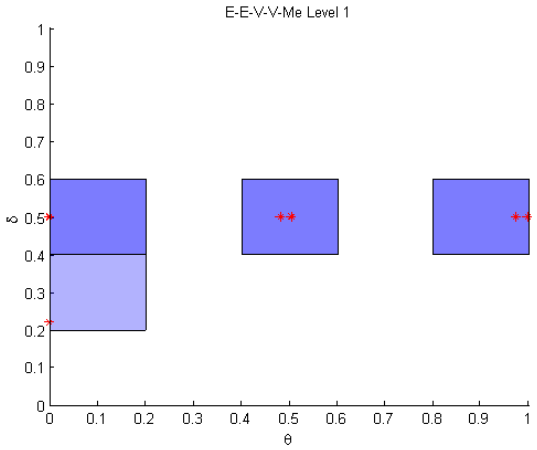



Fig. 2.21: Projection on $\theta, \delta$ of boxes and solutions after pruning level 1.

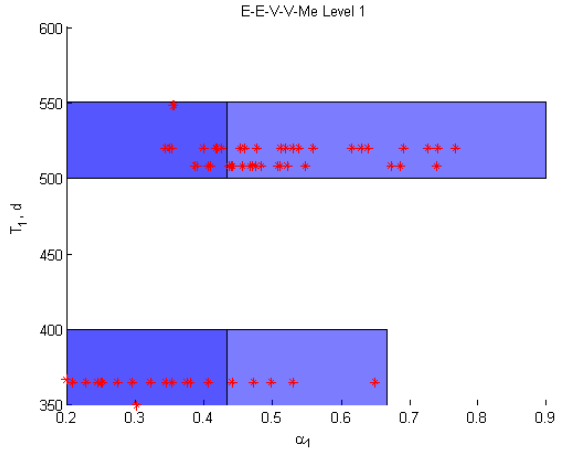



Fig. 2.22: Projection on $\alpha_1, T_1$ of boxes and solutions after pruning level 1.

The search at level 2 reveals that the solutions are no more spread along $t_0$ (Fig. 2.23): thus, the reduced size of the boxes allows identifying a few launch windows, between 4900 and 5200 MJD2000 in particular. This result was expected and justifies choice for a smaller box size along $t_0$ after level 1. In Fig. 2.24 it is noticeable that the time of flight $T_2$ for the E-V leg should be around 430 d. The projection of the boxes along the axes of the Earth swing-by, Fig. 2.25, shows that the ideal Earth swing-by angle $\gamma_1$ is around 0. No pruning is done on $r_{p,1}$, as the solutions are spread in the whole span.

The incremental approach proceeds in the same way up to level 4. At this point, the global solutions are found. Table 2.12 shows the comparison of the incremental approach with the two all-at-once approaches, in terms of objective function and number of function evaluations.

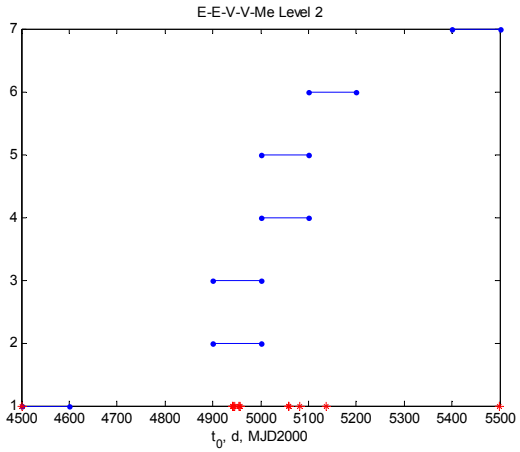



Fig. 2.23: Projection on $t_0$ of boxes and solutions after pruning level 2.

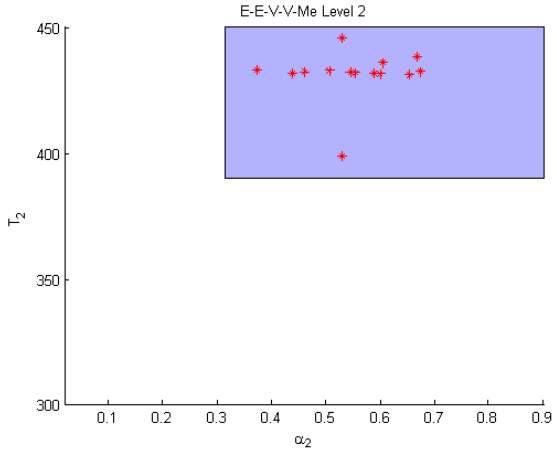



Fig. 2.24: Projection on $\alpha_2, T_2$ of boxes and solutions after pruning level 2.

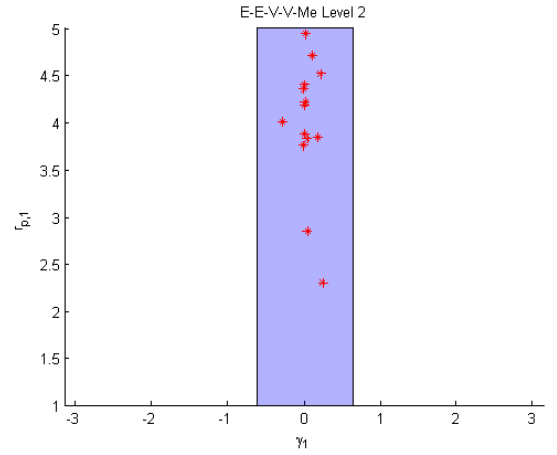



Fig. 2.25: Projection on $\gamma_1, r_{p,1}$ of boxes and solutions after pruning level 2.

Table 2.12: E-E-V-V-Me results for 4 different approaches, values computed on 20 runs.

| | Avg. no. of obj. fun. eval. | Time for objective function evaluation [s] | Best $\Delta v$ [km/s] Avg. | Std. dev. |
|---|---|---|---|---|
| DE all-at-once | 400010 | 5842 | 8.456 | 0.444 |
| PSO all-at-once | 460000 | 6900 | 6.094 | 0.920 |
| multi-start all-at-once | 427499 | 6412 | 4.599 | 0.865 |
| Incremental | 24397, 96674, 184340, 154754 | 3625 | 3.89 | 0.739 |

Table 2.13: Average time to evaluate the partial objective functions, for each level, in seconds.

| Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|
| $1.8 \cdot 10^{-3}$ | $3.5 \cdot 10^{-3}$ | $5.0 \cdot 10^{-3}$ | $1.5 \cdot 10^{-2}$ |

For this test case, the incremental algorithm outperforms all the other methods, using about the same number of function evaluations. Nevertheless, it has to be considered that all the all-at-once approaches evaluate the objective function for the complete

problem every time, while the incremental is evaluating that function only at level 4. The partial objective functions at lower levels are cheaper to compute, as they include less legs and zero-revolution Lambert problems, which are quicker to solve than the multi-revolution one at leg 4. Times for one function evaluation on an Intel Pentium 4 3 GHz are reported in Table 2.13. The result is that the total time spent in evaluating the objective function is far lower for the incremental approach than for the others. At the same time the incremental approach was able to identify better trajectories.
Fig. 2.26 plots the projection of the best trajectory found by the incremental algorithm during one of the 20 runs. The total $f_1$ is 4.55 km/s with a relative velocity at Mercury of 8.2 km/s. Note that, the reason why the relative velocity at Mercury is so high compared to the BepiColombo mission is that it was not included in any pruning criterion or in the objective function of the whole problem.

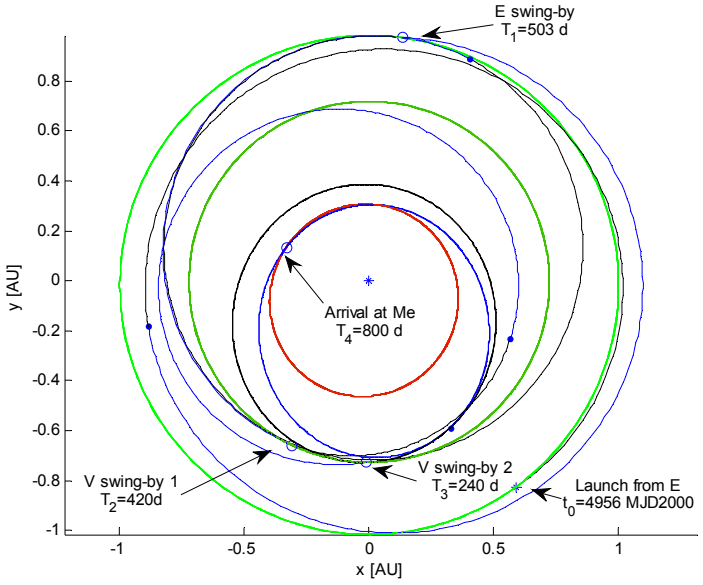



Fig. 2.26: Projection on the ecliptic plane of the best solution found by the incremental algorithm.

### 2.8.4 EVVMeMe Trasnfer

For this test we used the feasible set approach. The search was performed over and interval of 2000 days in the interval [3457, 5457] MJD2000. In this interval of launch dates there exist a particularly good solution for the EVVMeMe that was adopted as the chemical option for the ESA BepiColombo mission.
The search for a feasible set at each incremental level was performed with the software code EPIC. No pruning on the variables $\gamma$ and $r_p$ was considered and the number of resonant orbits was pre-assigned, i.e. fixed number of revolutions per leg.

The boundaries on the TOF for each leg were computed as a function of the number of revolutions. In particular given the pericentre $R_p$ and apocentre $R_a$ of the expected transfer orbit and the number of full revolutions, the lower bounds is the period of the elliptical orbit with pericentre $R_p$ and apocentre $R_a$ times the number of revolutions while the upper bounds is the period of the same orbit times the number of full revolutions plus one.

Special partial pruning criteria were used for the legs ending with Venus and Mercury. In particular for the arrival conditions at Venus the following special pruning criterion was used:

$$f_V = \frac{v_\theta^{\,2} + v_h^{\,2}}{v_r^{\,2}} + \Delta v_0 \tag{2.7}$$

and for the arrival conditions at Mercury the pruning criterion was:

$$f_i = \beta \frac{v_\theta^{\,2} + v_h^{\,2}}{v_r^{\,2}} + \sum_{k=1}^{i} \Delta v_k \tag{2.8}$$

The pruning functions were selected based on the required effect of the gravity assist manoeuvres. In particular Venus gravity manoeuvres are expected to maximize the change in the perihelion while the manoeuvres at Mercury, combined with the DSM, are supposed to minimize the relative velocity at Mercury.

The best sampled solution during the pruning process has a total Δv=6.5km/s with a $v_{inf}$=4km/s at Mercury and $v_{inf}$=3.869km/s at launch. This solution, represented in Fig. 54, is not a local minimum but just a sample in the pruned space. For comparison, the chemical option of the mission BepiColombo has a total Δv=4.08km/s with a $v_{inf}$=3.44km/s at Mercury and $v_{inf}$=3.762 km/s at launch.
The overall process required 225000 function evaluations and about 45 minutes of computational time on a Centrino 2GHz machine. The test was repeated 10 times with similar resulting pruned space.

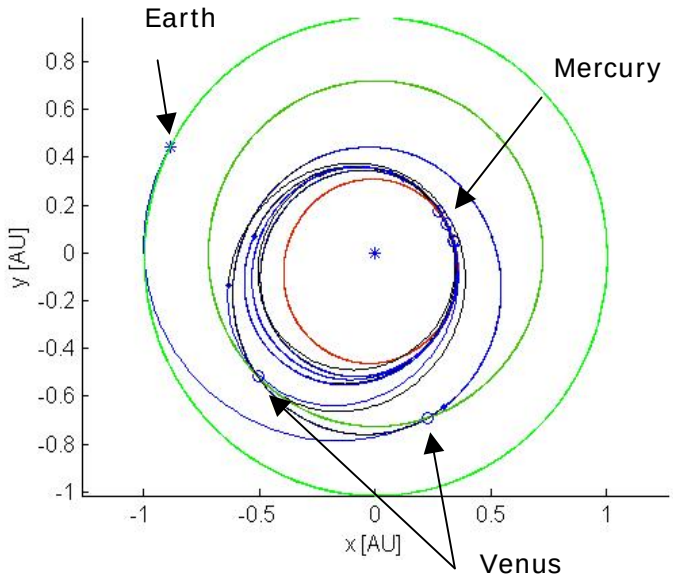



Fig. 2.27. Projection on the ecliptic plane of one of the sampled solutions in the pruned space

Fig. 2.28 to Fig. 2.38 are showing what remains after the pruning process. The red dots are the set of solutions computed by EPIC on the remaining part of the solutions space. Each red dot is not a local minimum but a feasible solution belonging to the feasible set, i.e. each one of the component of each solution belong to one of the remaining boxed after the pruning. In the same figures the green dot represents the BepiColombo

mission. The colour of the boxes is proportional to the average value of all the feasible solutions in the box. Red corresponds to lower Δv, blue to higher Δv values.

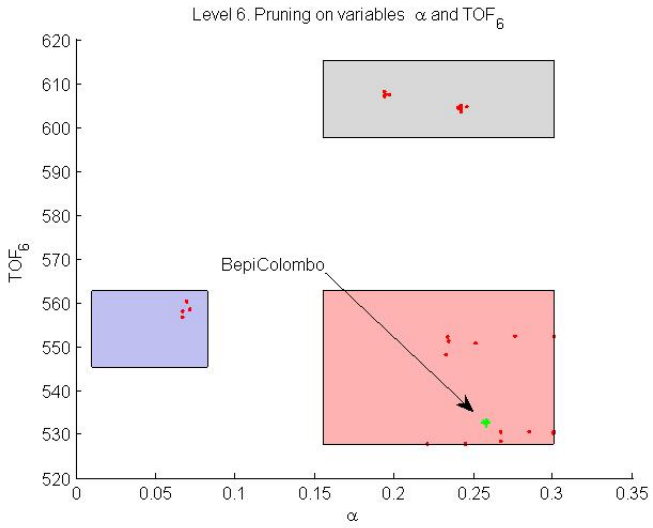



Fig. 2.28. Level 6 pruned space: variables α and TOF. The green point is the BepiColombo solution.

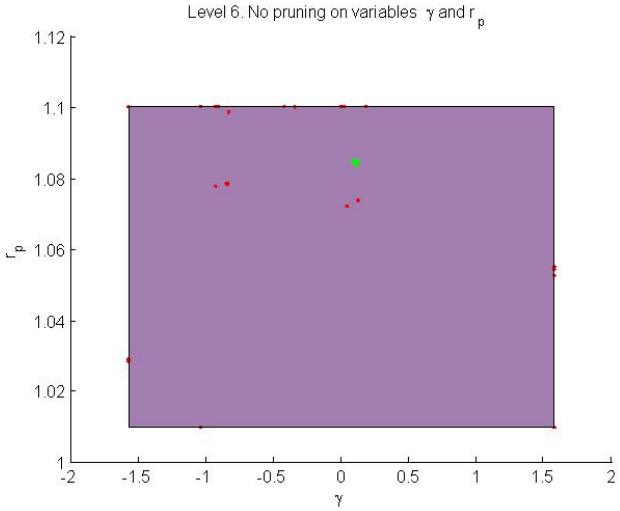



Fig. 2.29. Level 6 pruned space: variables γ and $r_p$. The green point is the BepiColombo solution.

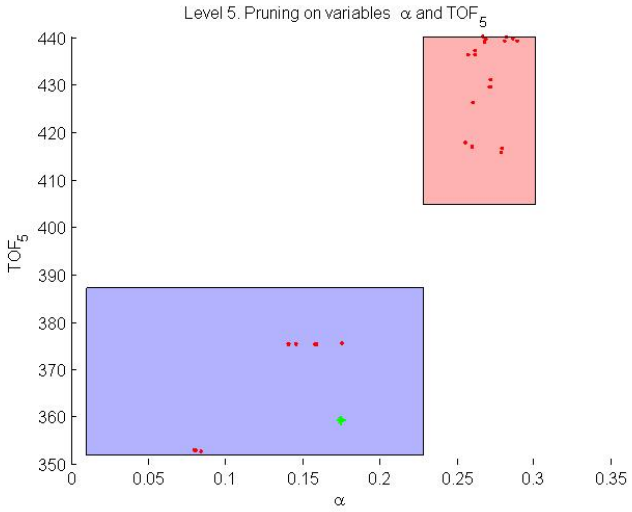



Fig. 2.30. Level 5 pruned space: variables α and TOF. The green point is the BepiColombo solution.

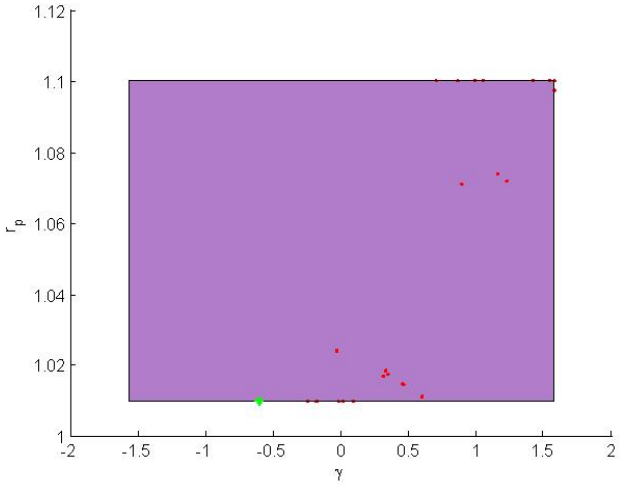



Fig. 2.31. Level 5 pruned space: variables γ and $r_p$. The green point is the BepiColombo solution.

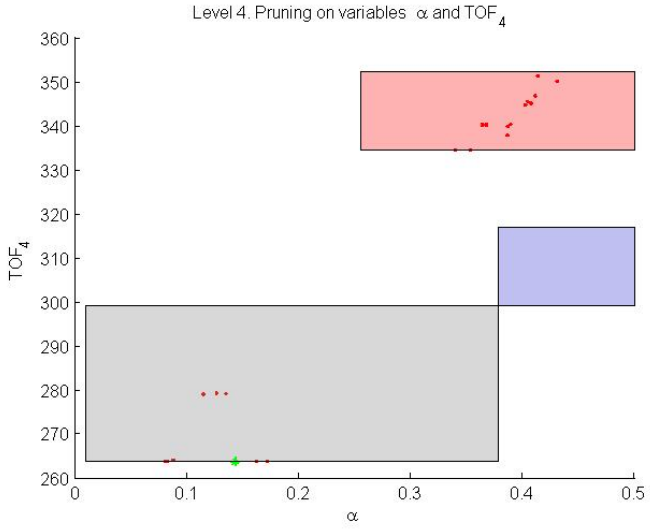



Fig. 2.32. Level 4 pruned space: variables α and TOF. The green point is the BepiColombo solution.

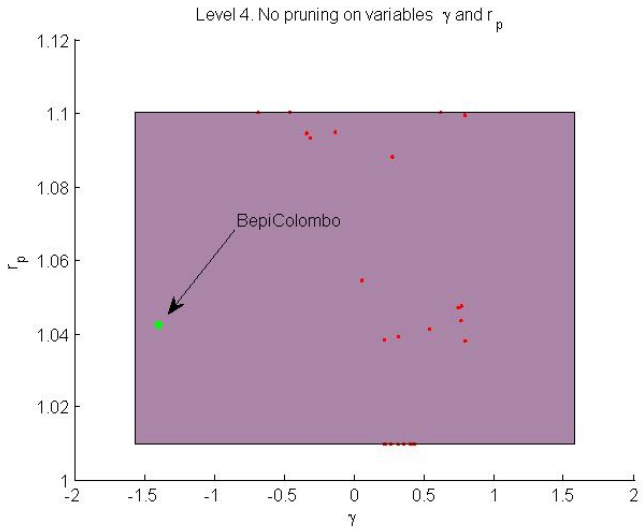



Fig. 2.33. Level 4 pruned space: variables $\gamma$ and $r_p$. The green point is the BepiColombo solution.

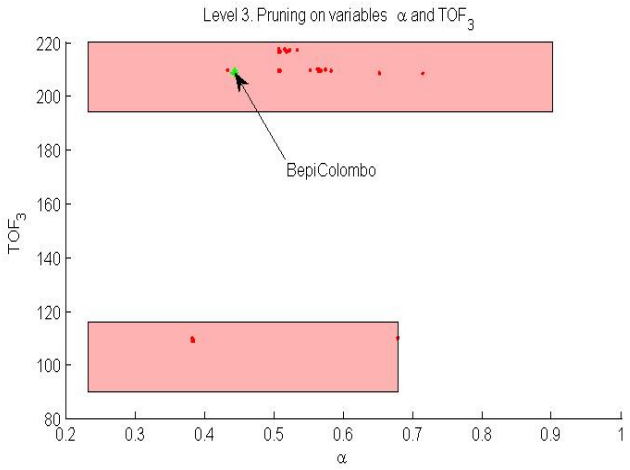



Fig. 2.34. Level 3 pruned space: variables $\alpha$ and TOF. The green point is the BepiColombo solution.

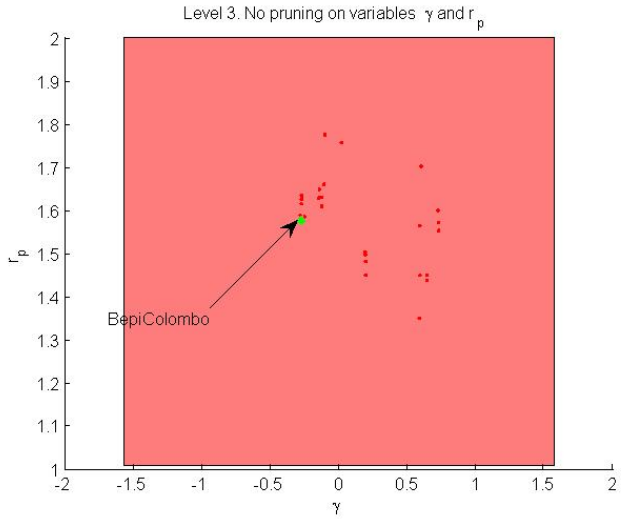



Fig. 2.35. Level 3 pruned space: variables $\gamma$ and $r_p$. The green point is the BepiColombo solution.

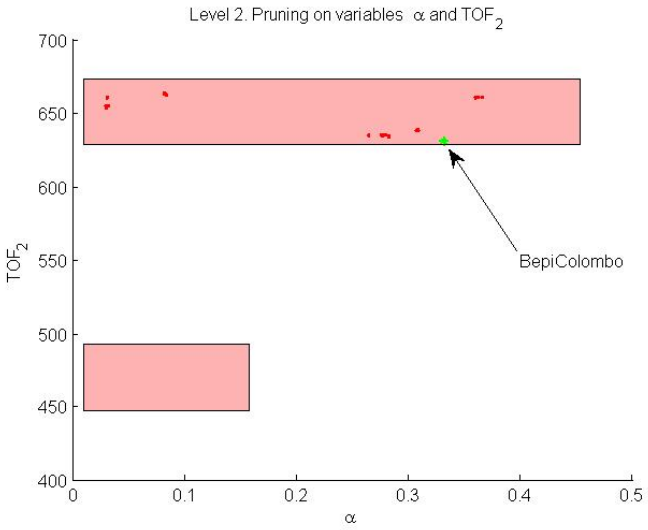



Fig. 2.36. Level 2 pruned space: variables α and TOF. The green point is the BepiColombo solution.

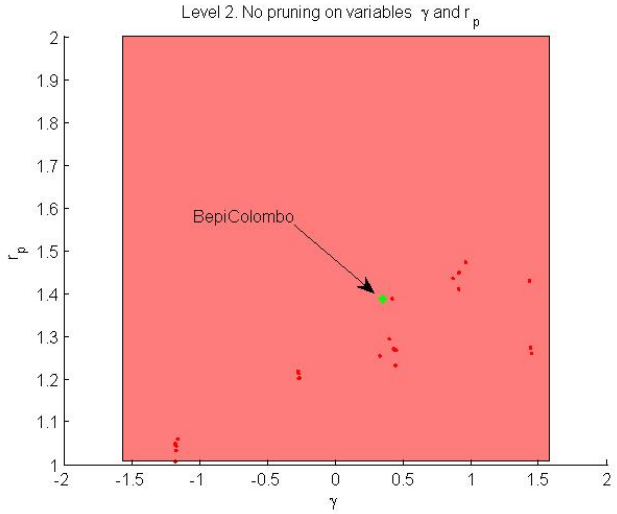



Fig. 2.37. Level 2 pruned space: variables γ and $r_p$. The green point is the BepiColombo solution.

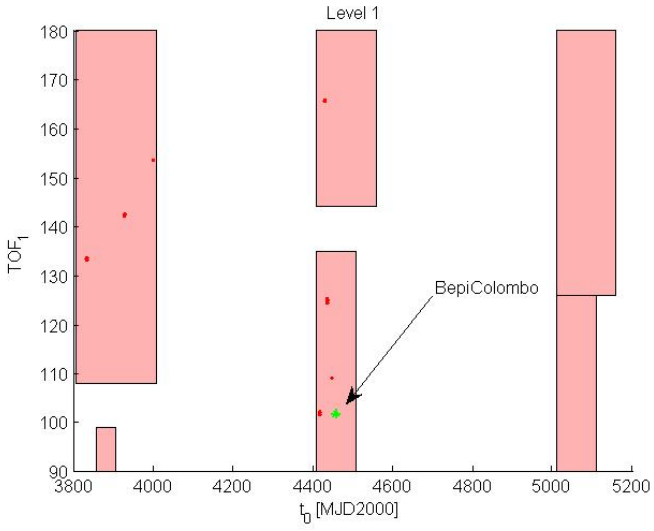



Fig. 2.38. Level 1 pruned space: variables α and TOF. The green point is the BepiColombo solution.

### *2.8.4.1 Search Space Analysis*

The importance of the selection of the right pruning criterion for each level can be understood by looking at the value of the total Δv at level 2 and 3 as a function of the parameter α and of the time of flight of the second leg of the EVVMeMeMe trajectory. All the other components of the solution vector correspond to the BepiColombo mission which is taken as the reference solution.

Fig. 2.39 to Fig. 2.48 shows the contour plots of the total Δv at level 2 as a function of the position of the DSM manoeuvre and the time of flight for the VV leg. Each contour plot was generated for a different value of the angle γ of the Venus gravity assist manoeuvre. The pruning threshold was fixed at 9 km/s. From the figures it can be seen that if the Δv was selected as pruning criterion three local minima could be identified, two of which, for α larger than 0.2 appear and disappear depending on the value of γ and one, a strong minimum, exist for every value of the angle γ. The local minimum visible in Fig. 2.42 for a time of flight of 630 days corresponds to the BepiColombo solution. However up to the second Venus flyby included, there is no way to distinguish among the three minima without an additional criterion.

Fig. 2.49 show the total Δv at the end of the third level with the spacecraft reaches Mercury for the first time. As can be seen the only part of the solution space that survives corresponds to the local minimum at 630 days, the BepiColombo solution.

At level 2, however, the required Δv at level 3 is know known and cannot be computed and a different pruning criterion is required to identify the correct portion of the solution space.

Alternatively, all the three regions including the three minima have to be preserved until the next level. At level three, the back pruning process would successfully remove the two suboptimal regions.

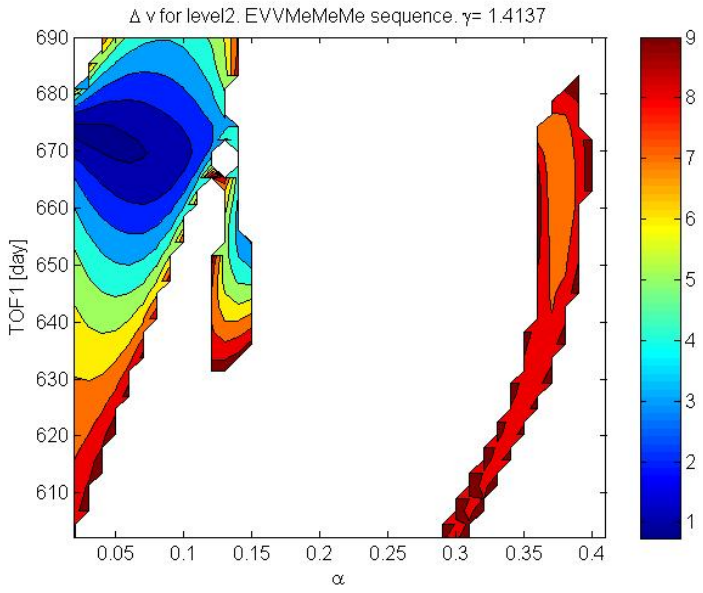



Fig. 2.39. Δv at level 2 for different TOFs and angle α

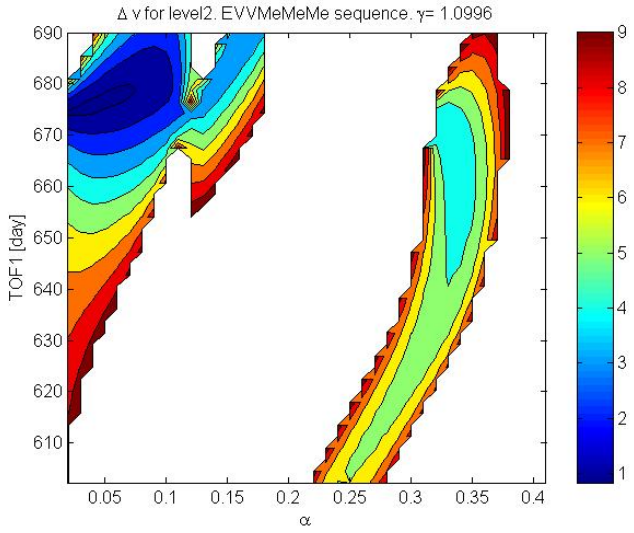



Fig. 2.40. Δv at level 2 for different TOFs and angle α

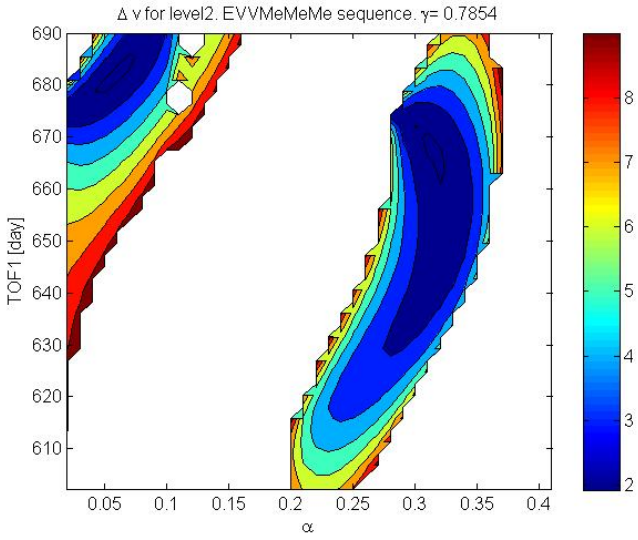



Fig. 2.41. Δv at level 2 for different TOFs and angle α

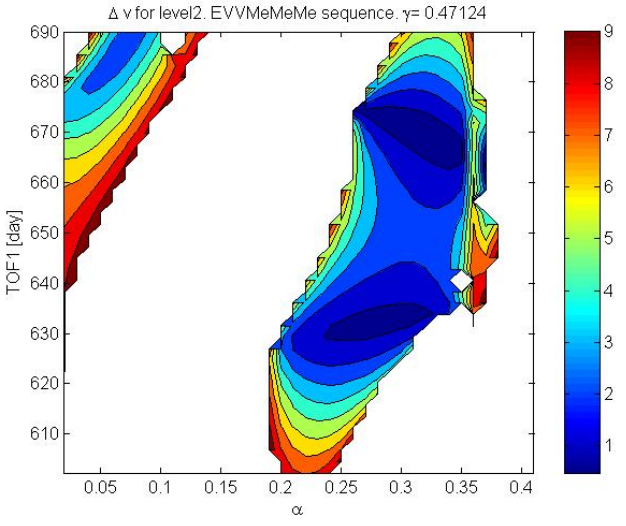



Fig. 2.42. Δv at level 2 for different TOFs and angle α

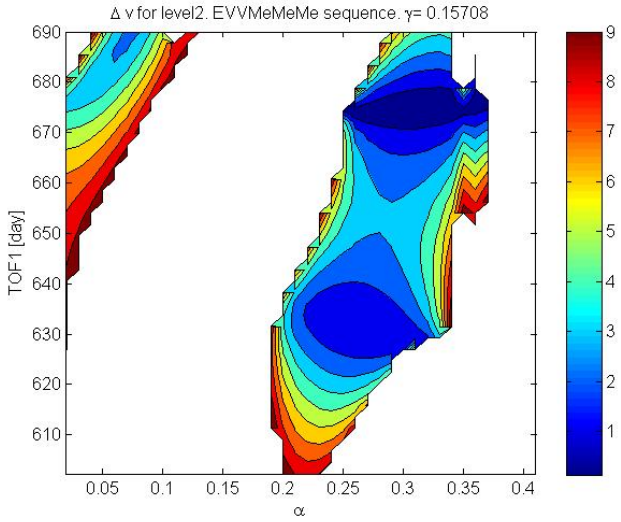



Fig. 2.43. Δv at level 2 for different TOFs and angle α

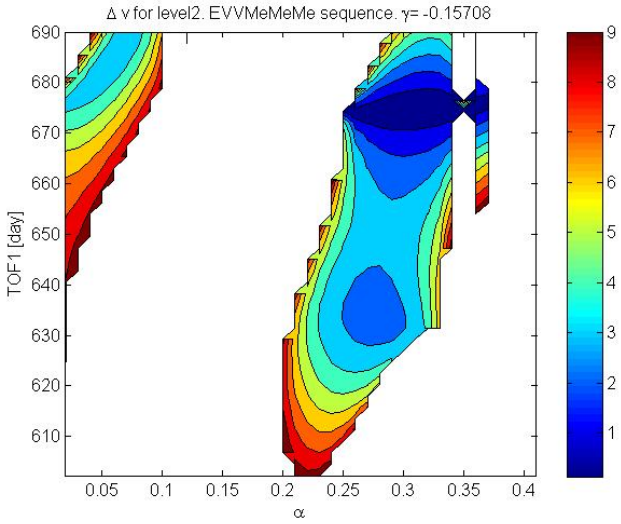



Fig. 2.44. Δv at level 2 for different TOFs and angle α

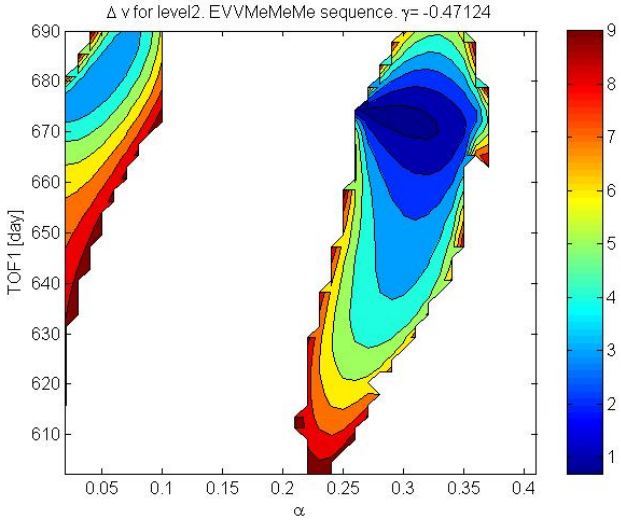



Fig. 2.45. Δv at level 2 for different TOFs and angle α

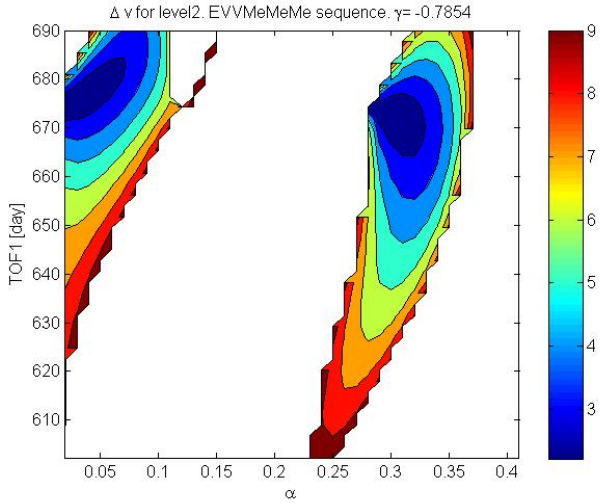



Fig. 2.46. Δv at level 2 for different TOFs and angle α

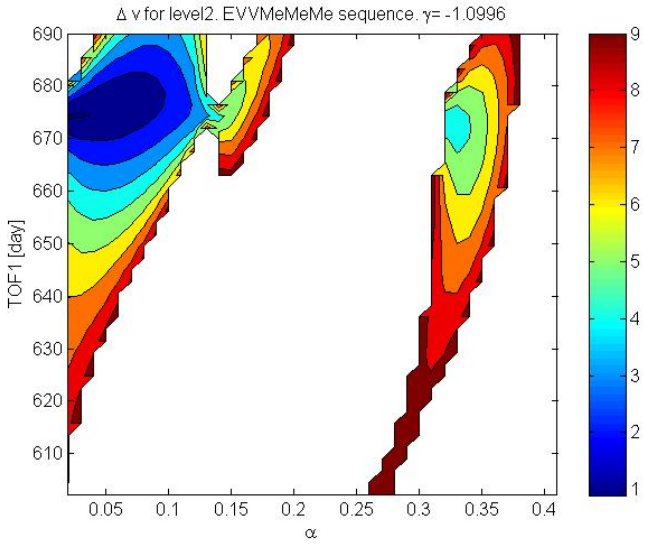



Fig. 2.47. Δv at level 2 for different TOFs and angle α

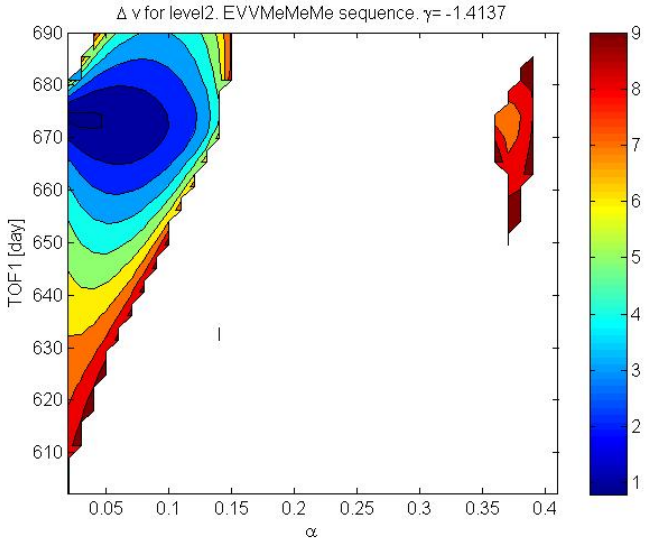



Fig. 2.48. Δv at level 2 for different TOFs and angle α

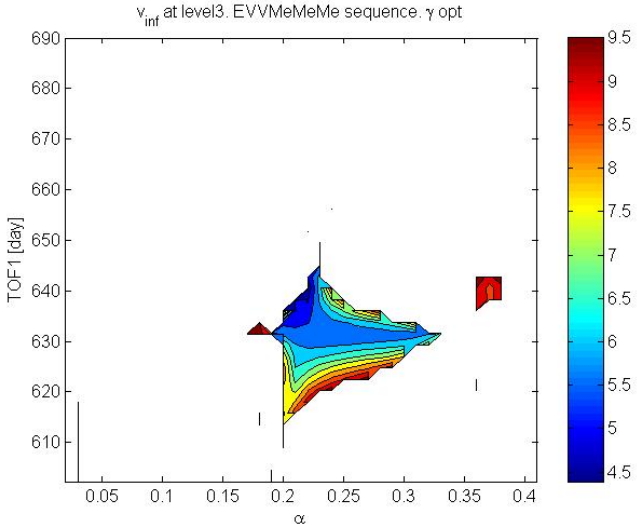



Fig. 2.49. Δv at level 3 for different TOFs and angle α

## 2.9 Final Remarks

In Part 2 of this report we presented an incremental approach to the solution of multiple gravity assist trajectories with deep space manoeuvres and an incremental approach to the automatic selection of a sequence of swing-bys.

Different variants of the main idea were presented together with their main advantages and drawbacks.

In particular, through the introduction of an affine transformation, we managed to avoid the exponential growth of the number of possible paths although the structural complexity of the solution spaces increases. The affine transformation, if fact, allows us to collect all the boxes that at each level survived the pruning process but introduces discontinuity and artificial local minima.

From the preliminary tests performed in this work we did not notice any significant increase in the difficulty of the search for a global minima due to the affine transformation itself. The actual difficulty arises from the collection of boxes rather than from the introduction of artificial nasty features in the solution space.

In fact, collecting the boxes does not allow the identification of isolated families of solutions.

On the other hand, preserving the families at each level would potentially result in an exponential growth of the number of families.

A possible solution could come from the back pruning. The backward pruning, in fact, reduces drastically the number of paths at level *i*-th thanks to the information collected at level *i+1*.

The backward pruning solves partially even the problem related to the correct use of the pruning function at each level. As we explained before, a pruning function directly related to the objective function of he entire trajectory can be totally inappropriate to prune the search space at each level *i*. On the other hand if the forward pruning at level I is not sufficient to discriminate between good areas and useless areas the backward pruning from level *i+1* seems to be quite effective in many practical cases.

An open problem is, therefore, to remove the affine transformation and some of the special pruning functions introduced in this work and to let the backward pruning do the job.

## 2.10 References


1. D. R. Myatt, V. M. Becerra, S. J. Nasuto, J. M. Bishop, "Advanced global optimisation for mission analysis and design", European Space Agency, Advanced Concepts Team, Ariadna Final Report 03-4101a, 2004
2. A. E. Petropoulos, J. M. Longuski, E. P. Bonfiglio, "Trajectories to Jupiter via gravity assists from Venus, Earth, and Mars", *Journal of Spacecraft and Rockets*, vol. 37, n. 6, p. 776-783, 2000
3. D. R. Jones, C. D. Perttunen, B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant", *Journal of Optimization Theory and Applications*, vol. 79, n. 1, p. 157-181, 1993
4. W. Huyer, A. Neumaier, "Global optimization by multilevel coordinate search", *Journal of Global Optimization*, vol. 14, n. 4, p. 331-355, 1999
5. R. Storn, K. Price, "Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces", *Journal of Global Optimization*, vol. 11, p. 341-359, 1997
6. J. Kennedy, R. Eberhart, "Particle swarm optimization", in *Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia, 1995
7. J. V. McAdams, D. W. Dunham, R. W. Farquhar, A. H. Taylor, B. G. Williams, "Trajectory design and maneuver strategy for the MESSENGER mission to Mercury", *Journal of Spacecraft and Rockets*, vol. 43, n. 5, p. 1054-1064, 2006
8. D. Garcia, P. De Pascale, R. Jehn, S. Campagnola, C. Corral, et al., "BepiColombo Mercury cornerstone consolidated report on mission analysis", ESA-ESOC Mission Analysis Office, MAO Working Paper No. 466, Darmstadt, 2006

# PART 3 EXTENDING THE GASP METHOD

## 3.1 Global Optimisation Algorithms

### 3.1.1 Introduction

This section presents a selection of global optimisation algorithms that have been employed throughout the project.

The algorithms described in this section are Differential Evolution (DE) and Particle Swarm Optimisation (PSO). Each of these optimisers can be considered an instance-based stochastic algorithm as they do not contain any probabilistic data of the model. Both algorithms have been tested on spacecraft trajectory optimisation problems, as reported in [1].

In the following sections each algorithm is described briefly. For more comprehensive descriptions please refer to the relevant references provided.

### 3.1.2 Differential Evolution, DE

Differential Evolution is a novel incomplete probabilistic global optimiser based on Genetic Algorithms, and was the highest ranked GA-type algorithm in the First International Contest on Evolutionary Computation. Additionally, DE is the standard global optimisation algorithm implemented in *Mathematica*. The algorithm was developed by Rainer and Storn [2]. An overview of the algorithm is given below.

*Differential Evolution Algorithm (DE1)*

1. Initialisation: select population vectors uniformly randomly over search space
2. Repeat
3. For each population vector $\mathbf{x}_i$
4. Select two other individuals uniformly randomly over entire population, $\mathbf{x}_2$ and $\mathbf{x}_3$
5. Create test vector $\mathbf{x}'_i = \mathbf{x}_i + F(\mathbf{x}_3 - \mathbf{x}_2)$
6. Evaluate $f'_i$.
7. If $(f'_i < f_i)$
8. Replace $\mathbf{x}_i$ with $\mathbf{x}'_i$
9. end
10. end.
11. Until Convergence

In this work we have used a Matlab implementation of Differential Evolution which can be found at:

http://www.icsi.berkeley.edu/~storn/devec3.m

### 3.1.3 Particle Swarm Optimisation, PSO

PSO was originally designed as a simulation of flocking behaviour in birds, although its potential for optimisation was recognised shortly afterwards. Each particle, analogous to the idea of an individual in genetic algorithms, has a position within the search space and a velocity, both of which are initialised randomly.

The algorithm was first presented by Kennedy and Eberhart in 1995 [3]. In the original form of the algorithm, each particle keeps tracks of the position of the best solution it has so far encountered, and also knows the *globally* best solution found by the entire population. The velocity is updated by two main components: the *cognitive* component, which attracts the particle towards its own best solution, and the *social* component, which attracts the particle to the best known solution. The essence of the algorithm is encapsulated in the following equations:

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \tag{3.1}$$

$$\mathbf{v}_i = \omega \mathbf{v}_i + \eta_1 r_1 (\mathbf{x}^\star_p - \mathbf{x}_i) + \eta_2 r_2 (\mathbf{x}^\star_g - \mathbf{x}_i), \tag{3.2}$$

where $\mathbf{x}^\star_p$ is the individual best solution of the $i^{th}$ particle, $\mathbf{x}^\star_g$ is the globally best known solution, and $r_1$, $r_2$ are uniform random numbers in the interval [0,1]. An overview of the algorithm is given below.

*Particle Swarm Optimisation Algorithm*

1. Initialise $\mathbf{x}$ uniformly randomly over search space.
2. Initialise $\mathbf{v}$ uniformly randomly within hyperparallelipiped of scale $v_{max}$ of the search space.
3. Repeat
4. For each population vector $\mathbf{x}_i$
5. Calculate objective function $f_i$
6. If $(f_i < f^\star_{pi})$
7. $f^\star_{pi} = f_i$
8. $\mathbf{x}^\star_{pi} = \mathbf{x}_i$
9. end
10. If $(f_i < f^\star_g)$
11. $f^\star_g = f_i$
12. $\mathbf{x}^\star_g = \mathbf{x}_i$
13. end
14. $\mathbf{v}_i = \omega\mathbf{v}_i + \eta_1 r_1(\mathbf{x}^\star_p - \mathbf{x}_i) + \eta_2 r_2(\mathbf{x}^\star_g - \mathbf{x}_i)$
15. $\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i$
16. end
17. Until Convergence

In this work we have used a C implementation of PSO known as Standard PSO 2006, which can be found at:

http://www.particleswarm.info/Standard_PSO_2006.c.

This PSO implementation has been interfaced to Matlab by means of a MEX function.

## 3.2 Pruning Algorithm for Model 2

### 3.2.1 Introduction

A pruning method known as GASP (Gravity Assist Space Pruning) has been proposed [1],[4] to solve Multiple Gravity Assist (MGA) trajectories with a known planetary sequence and no deep space manoeuvres. GASP showed that the vast majority of such a search space consists of infeasible, or very undesirable, solutions. This observation motivated the development of a method for producing reduced search spaces by pruning, thus allowing global optimisation techniques to be applied more successfully on the reduced domains. Secondly this method took advantage of the sequential nature of the problem. By pruning on a phase by phase basis, large savings in computational time were achieved.

Designing multiple gravity assist missions with no deep-space manoeuvres is limited in scope, since many possible trajectories cannot be considered, and, as practice shows, deep space manoeuvres are used in real missions. If the problem of multiple gravity assist with deep space manoeuvres could be pruned efficiently, then the computational cost of optimizing such trajectories may be significantly

reduced. The introduction of deep space manoeuvres offers the further advantage of providing a reasonable approximation of multiple gravity assist trajectories with low-thrust arcs. If a transfer arc is no more simply ballistic but is shaped by one or more propelled manoeuvres (either impulsive or low-thrust) the number of degrees of freedom increases significantly. Hence, an efficient solution process would have to make use of additional information to reduce the number of possible alternatives (pruning the search space) so reducing the computational cost, and increasing the likelihood of finding good solutions.

A method is now described that is capable pruning the search space of missions with multiple gravity assist trajectories that contain deep space manoeuvres for the particular case of powered swingby's. The proposed pruning technique can be seen as an extension of the original GASP algorithm.

### 3.2.2 Desirable Properties

GASP was efficient and desirable for many reasons. In an attempt to extend its functionality the following properties of the original algorithm can be noted:

1. The algorithm used the sequential nature of the problem;
2. it prunes on the basis of feasibility;
3. and it produces families of feasible solutions

As the trajectory model with deep space manoeuvres and powered gravity assists that is available also exhibits sequential properties, it is a good idea to take advantage of this. In this manner, a mission that contains multiple phases can initially be pruned one phase at a time.

GASP employed grid sampling as a method for evaluating the search space of each phase. If the sampled point satisfied a certain constraint, then the point was considered feasible and kept, else it was infeasible and hence removed. While grid sampling is usually considered inefficient, as only a two dimensional search was ever performed it was acceptable. It is shown in this report that grid sampling in the number of dimensions required when a deep space manoeuvre is being considered is not practical. An alternate method for sampling the search space is described in the next section.

Finally the ability to define *solution families* across the whole scope of the mission ensured that feasible regions in one phase were not combined with feasible regions in later phases, unless it was physically possible. This reduced the number of combinations of bounding boxes and prevented impossible trajectories that involved being in two places at one time from being generated.

### 3.2.3 Pruning Algorithm with Deep Space Manoeuvres

The following sections define the sequential optimisation problem being addressed, and describe how the algorithm works in a step by step manner. It should be

noted that the pruned search space will still contain many local minima. For this reason a global optimisation algorithm is required after the pruning is complete. Several test cases including those based upon real missions have been implemented to demonstrate the effectiveness of the pruning strategy. Complexity analysis shows that the algorithm scales well as the number of phases in the mission increases. The pruning method presented here has been published during the course of this project at the 2007 IEEE World Congress on Evolutionary Computation, Singapore [5].

### 3.2.4 Problem Definition

The problem of interest may be formulated as a multi-stage optimisation problem (MSOP). An example of how the original GASP algorithm may be case as a multi-stage optimisation problem is given below.
*MSOP:* Find the decision vector

$$\mathbf{x} = \left[\mathbf{x}_1^T, \quad \mathbf{x}_2^T, \quad \cdots, \quad \mathbf{x}_{s+1}^T\right]^T \in \Omega \tag{3.3}$$

where each sub-vector $\mathbf{x}_k$, $k = 1, \ldots, s+1$ is associated with a stage of the problem, to minimise the objective function

$$f(\mathbf{x}, \mathbf{z}_1, \cdots, \mathbf{z}_{s+1}) \tag{3.4}$$

subject to the following constraints

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_1, \cdots, \mathbf{x}_{k+1}), k = 1, \cdots, s \tag{3.5}$$

$$\mathbf{g}_k(\mathbf{z}_k) \leq 0, k = 1, \cdots s+1 \tag{3.6}$$

where for example the inequality constraints may represent bounds on individual $\Delta V$'s. If we let $\Omega$ be the Cartesian product of all the s+1 hyper-rectangles, then $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_{s+1}$. Each hyper-rectangle is bounded such that $\Omega_k = \{\mathbf{x}_k \in \mathrm{R}^{n_k} | \mathbf{x}_L^{(k)} \leq \mathbf{x}_k \leq \mathbf{x}_U^{(k)}\}, k = 1 \ldots, s+1$. In this manner $\Omega$ defines the whole search space for a single mission and likewise $\Omega_k$ defines the search space for a single phase of a mission.

The objective function is assumed to be scalar as in almost every case it is the sum of all $\Delta V$'s. Such a mapping can be written as $f : \Omega \times \mathrm{R}^{q_1} \times \cdots \times \mathrm{R}^{q_{s+1}} \rightarrow \mathrm{R}$.

The vectors $\mathbf{z}_k \in \mathrm{R}^{q_k}$, $k$=1,...,$s+1$, are intermediate variables associated with each stage. For example, they may represent $\Delta V$'s associated with different manoeuvres.

Each of the intermediate functions $\mathbf{h}_k : \Omega_1 \times \Omega_2 \times \cdots \times \Omega_{k+1} \rightarrow \mathrm{R}^{m_k}$ and $\mathbf{g}_k : \mathrm{R}^{q_k} \rightarrow \mathrm{R}^{d_k}$ is associated with a particular stage $k$.

Note that the calculation of the constraint function $\mathbf{g}_k$ depends on intermediate variables calculated at stage $k$, and the objective function depends on the values of the intermediate variables $\mathbf{z}_k$, $k$=1,..., $s+1$, hence a specific order must be followed to evaluate the objective function $f$ and the constraint functions $\mathbf{g}_k$.

The presence of inequality constraints in the MSOP requires careful considera-

tion. Although bounds on the decision variables are easy to manage, more general inequality constraints are more difficult to handle in global optimisation. A plausible method is to prune the search space based on feasibility (i.e. constraint satisfaction). This has an important benefit: the size of the search space is reduced hence simplifying the optimisation task. One simple method of pruning is to grid sample the search space with a suitable resolution so that unfeasible areas can be detected by evaluating the constraint functions, and subsequently eliminated, leaving a reduced search space where optimisation can be applied. However, the cost of grid sampling with reasonable resolutions may be prohibitive when the search space dimension is larger than two or three.

The mission considered by the original GASP method includes powered gravity assist at intermediate planets, and if required a braking manoeuvre at the arrival planet for orbit insertion. The problem addressed by GASP can be cast as a MSOP. Here, the decision vector $\mathbf{x}$ consists of the launch date and transfer times between planets, the intermediate variables $\mathbf{z}_k$ are the $\Delta\mathbf{v}$'s applied at each planet. The functions $\mathbf{h}_k$ represent the calculations that are required to find the intermediate variables (solution of Lambert problems, swing-by models), the objective function is the sum of the magnitudes of the $\Delta\mathbf{v}$'s. The constraint functions $\mathbf{g}_k$ are related to upper bounds on the $\Delta\mathbf{v}$'s at each planet (as thrusters have limits), as well as lower bounds for the periapsis radius at each swing-by planet (to keep a safe distance from the planet).

The following sections describe the pruning algorithm with deep space manoeuvres. A simple two phase mission with a deep space manoeuvre in each phase and a swingby followed by a breaking manoeuvre will be used as an example. This method can easily be extended to incorporate multiple deep space manoeuvres and more phases.

### 3.2.5 Two Phase Mission with Deep Space Manoeuvres

A hypothetical mission is considered for the purposes of describing the pruning method. The mission consists of two phases, each containing a deep space manoeuvre. The phases are linked with a powered swingby. The constraints that determine feasibility are the launch velocity (modeled as an impulsive manoeuvre), the impulsive deep space manoeuvres and a breaking manoeuvre to achieve orbital insertion.

The notation used to describe the algorithm is as follows. The super-script in parenthesis (e.g. $^{(1)}$, $^{(2)}$) indicates the phase of the mission. $\Delta v_{\text{dep}}$ is the impulsive manoeuvre at departure, $\Delta v_{\text{DSM}}$ is an impulsive manoeuvre at deep space, $\Delta v_{\text{ga}}$ is an impulsive manoeuvre at a gravity assist planet, $\Delta v_{\text{b}}$ is a braking manoeuvre that is performed for orbit insertion purposes, $\mathbf{v}_{\text{in}}$ is the arrival velocity at a swingby planet, $\mathbf{v}_{\text{out}}$ is the departure velocity from a swingby planet, $t_0$ is the mission launch date, $t_{\text{arr}}$ is an arrival time at the end of one phase, $t_{\text{dep}}$ is the departure time at the beginning of a phase, $T_{\text{of}}$ is the time of flight of a given phase.

*Model 2 Revisited*

The model that this pruning algorithm uses requires two timing parameters to be set for each phase of the mission. The initial time $t_0$ specified in MJD2000 and the time taken in days for the spacecraft to fly from the departure planet to

its destination, $t_{of}$. Each deep space manoeuvre is then characterized in polar coordinates by the following parameters:

- $r$: Distance from the Sun (km) at which the DSM occurs. The vector from the Sun to the deep space manoeuvre is denoted as $\mathbf{r}_{\text{DSM}}$.
- $\theta$: In-plane angle (angle between $\mathbf{r}_{p1}$ and the projection of $\mathbf{r}_{DSM}$ on the orbital plane of the first planet). This projection is denoted as $\mathbf{r}'_{\text{DSM}}$ in Figure 3.1.
- $\phi$: out of plane angle (angle between $\mathbf{r}_{\text{DSM}}$ and its projection $\mathbf{r}'_{\text{DSM}}$ on the orbital plane of the first planet)
- $\alpha$: The timing of the DSM as a fraction in the interval $[0, 1]$ of the time of flight.

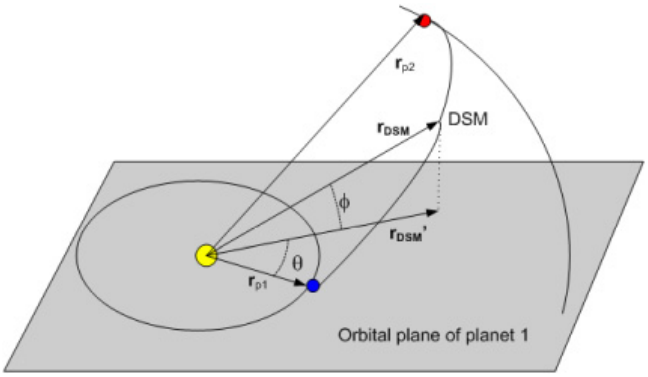



Figure 3.1: Graphical description of model 2. A single phase with 1 deep space manoeuvre.

One of the advantages of using this model is that it allows missions with multiple phases to be considered. In these cases any two consecutive deep space flight phases can be considered independently without taking into account the swingby of the planet. This is due to the fact that the initial velocity is not needed to compute the Lambert arc. Rather, the initial and final velocities are outputs from the Lambert solver. Once two consecutive legs are computed, both the incoming and outgoing velocity at the planet become available and the swingby (with the powered model) can be computed.

In this way, in order to create the whole trajectory, the only requirement is that the time at each planet is the same for all the phases arriving or departing from that planet. At this point, no constraints are considered on the incoming and outgoing velocities. Thus, it is possible to analyse (optimise, prune) the deep space flight phases first, then match them with the swingbys and prune again on the basis of the feasibility of the swingby.

*Algorithm 1: First Phase Pruning*

The decision vector associated with this leg is as follows:

$$\mathbf{x}_1 = [t_0,\, T_{of}^{(1)}, r^{(1)}, \alpha^{(1)}, \theta^{(1)}, \phi^{(1)}]^T \tag{3.7}$$

In cases when there are no DSM's present in a phase then the decision vector will comprise only the two timing variables ($t_0$, $T_{of}^{(1)}$). However there will be occasions when a single phase consists of multiple DSM's. In such a case the original parameters, $(r^{(1)}, \alpha^{(1)}, \theta^{(1)}, \phi^{(1)})$ will themselves become vectors of length $j$. Where $j$ represents the number of manouevres present in the phase.

Assume that $\mathbf{x}_1$ is initially limited to the hyper-rectangle $\Omega_1 \subset \Re^6$, such that each element of $\mathbf{x}_1$ is initially bounded between lower and upper limits: $x_{1,j}^l \leq x_{1,j} \leq x_{1,j}^u$, $j = 1, \ldots, 6$.

We are interested in pruning the search space by finding an estimate of the feasible regions of the search space associated with the first leg, with respect to constraints on the departure impulse $\Delta v_{\text{dep}}^{(1)}$, and the deep space manoeuvre impulse $\Delta v_{\text{DSM}}^{(1)}$. To find such an estimate, we propose the use of a local optimisation algorithm, which can be started from multiple random vectors within the admissible region $\Omega_1$, and stopped when feasible vectors are found. This procedure is then followed by the application of a clustering algorithm to find an estimate of the region. We have used in this work the mean shift clustering algorithm [6], [7], which does not require an a priori specification of the number of clusters. Thus the algorithm to prune the search space associated with the first phase is as follows:

1. Generate randomly $N_1$ starting vectors within the admissible region for the first phase: $\bar{\mathbf{x}}_1^i \in \Omega_1 \subset \Re^6$, $i = 1, \ldots, N_1$.

2. Start a constrained local optimisation algorithm, such as sequential quadratic programming, from each initial vector $\bar{\mathbf{x}}_1^i$, $i = 1, \ldots, N_1$ to solve the following problem:

$$\min_{\mathbf{x}_1} f_1(\mathbf{x}_1) = \Delta v_{\text{dep}}^{(1)} + \Delta v_{\text{DSM}}^{(1)} \tag{3.8}$$

subject to

$$\begin{aligned} \begin{bmatrix} \Delta v_{\text{dep}}^{(1)} \\ \Delta v_{\text{DSM}}^{(1)} \end{bmatrix} &= \mathbf{h}_1(\mathbf{x}_1) \\ \Delta v_{\text{dep}}^{(1)} &\leq \Delta v_{\text{dep}}^{\max} \\ \Delta v_{\text{DSM}}^{(1)} &\leq \Delta v_{\text{DSM}}^{\max} \end{aligned} \tag{3.9}$$

where $\Delta v_{\text{dep}}^{(1)}$ is the impulsive manoeuvre at launch, $\Delta v_{\text{DSM}}^{(1)}$ is the impulsive manoeuvre performed at deep space during phase 1. Note that the constrained optimisation algorithm may be stopped as soon as a feasible vector satisfying the inequality constraints is found. If a feasible vector is found, it is recorded as $\hat{\mathbf{x}}_1^i$. If no feasible vector is found starting from $\bar{\mathbf{x}}_1^i$, then the optimisation starts again with the next value of $i$. This step ends with a collection of feasible vectors $\hat{\mathbf{x}}_1^i$ $i = 1, \ldots, M_1$, where $M_1 \leq N_1$. If no feasible vectors are found, the algorithm stops.

3. Given the set of feasible vectors found in step 2, run the clustering algorithm to find a set of $P_1$ clusters, so that each vector $\hat{\mathbf{x}}_1^i$ is assigned to a cluster $C_j$, where $i = 1, \ldots, M_1$ and $j = 1, \ldots, P_1$.

4. This step uses the information in the clusters to form one bounding box for each cluster $C_j$, $j = 1, \ldots, P_1$. Let $\mathbf{x}^{\min,j} \in \Re^6$ be a vector so that each of its elements $x_i^{\min,j}$ is the minimum i-th coordinate value for all the vectors in cluster $C_j$. Similarly, let $\mathbf{x}^{\max,j} \in \Re^6$ be a vector so that each of its elements $x_i^{\max,j}$ is the maximum i-th coordinate value for all the vectors in cluster $C_j$. Then the bounding box for cluster $j$ is defined as

$$B_j^{(1)} = \{\mathbf{x} \in \Omega_1 \subset \Re^6 | \, \mathbf{x}^{\min,j} \leq \mathbf{x} \leq \mathbf{x}^{\max,j}\}. \tag{3.10}$$

Note that each $B_j^{(1)}$ is a subset of $\Omega_1$, the original search space for the decision variables associated with phase 1. The set of bounding boxes $B_j^{(1)}$, $j = 1, \ldots, P_1$ represents the initially pruned search space for phase 1 (this set is updated later, in a backward constraining step). Denote $\mathcal{B}^{(1)}$ as the set of bounding boxes $B_j$, $j = 1, \ldots, P_1$.

*Algorithm 2: Second Phase Pruning*

The model employed in this study is based on the patched conic approach. This requires that the time of arrival at the end of the first phase to be identical to the time of departure for the second phase. Given values for the launch time $t_0$ and the time of flight for the first leg $T_{\text{of}}^{(1)}$, the time of arrival at the end of phase 1, which is equal to the time of departure for phase 2, is given by:

$$t_{\text{arr}}^{(1)} = t_{\text{dep}}^{(2)} = t_0 + T_{\text{of}}^{(1)} \tag{3.11}$$

We have found in phase 1 a set of intervals of feasible values for the arrival time $t_{\text{arr}}^{(1)}$. Such intervals are derived from the first two co-ordinates of the bounding boxes for phase 1, $B_j^{(1)}$, $j = 1, \ldots, P_1$. Since $t_{\text{arr}}^{(1)} = t_{\text{dep}}^{(2)}$ then it only makes sense to consider values of $t_{\text{dep}}^{(2)}$ within the same intervals. This was the main idea that was exploited in the design of the original GASP method [1], [4]. Let us denote the feasible intervals for $t_{\text{dep}}^{(2)}$ as $\mathcal{I}_j, j = 1, \ldots, P_1$. Note that such intervals may, in general, overlap. Denote $\mathcal{I}$ as the union of all intervals $\mathcal{I}_j$, $j = 1, \ldots, P_1$.

Given that we are assuming a powered swingby, the arrival and departure velocities are decoupled (the departure velocity does not depend on the arrival velocity, provided any bound constraints on the $\Delta v$ magnitude are not hit), so we can compute the second phase without having computed first the powered swingby. Assume that the second phase also involves a single deep space manoeuvre, so that the decision vector for the second phase is:

$$\mathbf{x}_2 = [T_{\text{of}}^{(2)}, r^{(2)}, \alpha^{(2)}, \theta^{(2)}, \phi^{(2)}]^T \tag{3.12}$$

where $T_{\text{of}}^{(2)}$ represents the time of flight for the second leg, and $\{r^{(2)}, \alpha^{(2)}, \theta^{(2)}, \phi^{(2)}\}$ are parameters associated with the deep space manoeuvre that takes place in the second phase. Note that there is one less parameter here than in the launch phase. Let us denote the initial admissible region for $\mathbf{x}_2$ as $\Omega_2$.

In order to compute the second phase, it is necessary to specify values for $t_{\text{dep}}^{(2)}$ and $\mathbf{x}_2$. Define an augmented vector associated with the second phase as follows:

$$\mathbf{X}^{(2)} = [t_{\text{dep}}^{(2)}, \mathbf{x}_2^T]^T \in \Re^6 \tag{3.13}$$

Let us define the admissible region for this vector as $\bar{\Omega}_2 = \mathcal{I} \times \Omega_2$. The pruning algorithm for the second phase can now be described as follows:

1. Generate randomly $N_2$ starting vectors within the admissible region for the second phase: $\bar{\mathbf{X}}_2^i \in \bar{\Omega}_2 \subset \Re^6$, $i = 1, \ldots, N_2$.

2. Start a constrained local optimisation algorithm, such as sequential quadratic programming, from each initial vector $\bar{\mathbf{X}}_2^i$, $i = 1, \ldots, N_2$ to solve the following problem:
$$\min_{\mathbf{X}_2} f_2(\mathbf{X}_2) = \Delta v_{\text{DSM}}^{(2)} + \Delta v_{\text{b}}^{(2)} \tag{3.14}$$
subject to
$$\begin{aligned} \begin{bmatrix} \Delta v_{\text{DSM}}^{(2)} \\ \Delta v_b^{(2)} \end{bmatrix} &= \mathbf{h}_2(\mathbf{X}_2) \\ \Delta v_{\text{DSM}}^{(2)} &\leq \Delta v_{\text{DSM}}^{\max} \\ \Delta v_b^{(2)} &\leq \Delta v_b^{\max} \end{aligned} \tag{3.15}$$
where $\Delta v_{\text{DSM}}^{(2)}$ is the deep space manoeuvre, and $\Delta v_{\text{b}}^{(2)}$ is the braking manoeuvre at the final planet. Note that the constrained optimisation algorithm may be stopped as soon as a feasible vector satisfying the inequality constraint is found. If a feasible vector is found, it is recorded as $\hat{\mathbf{X}}_2^i$. If no feasible vector is found starting from $\bar{\mathbf{X}}_2^i$, then the optimisation starts again with the next value of $i$. This step ends with a collection of feasible vectors $\hat{\mathbf{X}}_2^i$ $i = 1, \ldots, M_2$, where $M_2 \leq N_2$. If no feasible vectors are found, the algorithm stops.

3. This step checks the feasibility of each of the vectors found in Step 2 with respect to the gravity assist manoeuvre. From each of the vectors found in Step 2, $\hat{\mathbf{X}}_2^i$ $i = 1, \ldots, M_2$, extract the departure time $t_{dep}^{(2,i)}$, and take the corresponding departure velocity vector $\mathbf{v}_{\text{out}}^{(2,i)}$ (which is computed as part of the evaluation of the second leg). Then, given values for $t_{dep}^{(2,i)}$, and $\mathbf{v}_{\text{out}}^{(2,i)}$, start a constrained local optimiser from $N_3$ randomly generated vectors $\mathbf{x}_1^j \in \mathcal{B}^{(1)}$, $j = 1, \ldots, N_3$, to solve the following problem:
$$\min_{\mathbf{x}_1} f_1(\mathbf{x}_1) = \Delta v_{\text{dep}}^{(1)} + \Delta v_{\text{DSM}}^{(1)} \tag{3.16}$$

subject to

$$\begin{aligned}
\mathbf{x}_1 &\in \mathcal{B}^{(1)} \\
\begin{bmatrix} \Delta v^{(1)}_{\text{dep}} \\ \Delta v^{(1)}_{\text{DSM}} \\ \mathbf{v}^{(1)}_{\text{in}} \end{bmatrix} &= \bar{\mathbf{h}}_1(\mathbf{x}_1) \\
\Delta v^{(1)}_{ga} &= q_1(\mathbf{v}^{(1)}_{\text{in}}, \mathbf{v}^{(2,i)}_{\text{out}}, r^{(1)}_{\min}) \\
t_0 + T^{(1)}_{\text{of}} - t^{(2,i)}_{\text{dep}} &= 0 \\
\Delta v^{(1)}_{\text{dep}} &\leq \Delta v^{\max}_{\text{dep}} \\
\Delta v^{(1)}_{\text{DSM}} &\leq \Delta v^{\max}_{\text{DSM}} \\
\Delta v^{(1)}_{\text{ga}} &\leq \Delta v^{\max}_{\text{ga}}
\end{aligned} \tag{3.17}$$

where $\mathbf{v}^{(1)}_{\text{in}}$ is the arrival velocity at the gravity assist planet, $\Delta v^{(1)}_{\text{ga}}$ is the impulsive manoeuvre performed at the gravity assist planet, $r^{(1)}_{\min}$ is the minimum allowed pericenter altitude during the gravity assist manoeuvre. If a feasible solution $\tilde{\mathbf{x}}_1$ is found out of the $N_3$ local optimiser runs, then $\tilde{\mathbf{x}}_1$ is stored, and the vector $\hat{\mathbf{X}}^i_2$ is confirmed as feasible, otherwise $\hat{\mathbf{X}}^i_2$ is discarded. This results in $Q_2 \leq M_2$ feasible vectors associated with the second phase, and $Q_1 \leq M_1$ feasible vectors associated with the first phase.

4. Given the set of feasible vectors found in step 3, run the clustering algorithm to find a set of $P_2$ clusters, so that each vector $\hat{\mathbf{X}}^i_2$, is assigned to a cluster $C^{(2)}_j$, where $i = 1, \ldots, Q_2$ and $j = 1, \ldots, P_2$.

5. This step uses the information in the clusters found in step 4 to form one bounding box for each cluster $C^{(2)}_j$, $j = 1, \ldots, P^{(2)}$. Let $\mathbf{X}^{\min,j} \in \Re^6$ be a vector so that each of its elements $X^{\min,j}_i$ is the minimum $i$-th coordinate value for all the vectors in cluster $C^{(2)}_j$. Similarly, let $\mathbf{X}^{\max,j} \in \Re^6$ be a vector so that each of its elements $X^{\max,j}_i$ is the maximum $i$-th coordinate value for all the vectors in cluster $C^{(2)}_j$. Then the bounding box for cluster $j$ is defined as

$$\bar{B}^{(2)}_j = \{\mathbf{X} \in \bar{\Omega}_2 \subset \Re^6 | \, \mathbf{X}^{\min,j} \leq \mathbf{x} \leq \mathbf{X}^{\max,j}\}. \tag{3.18}$$

Note that the bounding boxes found in step 5 are associated with the augmented variable $\mathbf{X}_2$ defined in equation (3.13). It is straightforward to find the bounding boxes $B^{(2)}_j$ that correspond to the original variable vector $\mathbf{x}_2$. Denote the set of such boxes as $\mathcal{B}^{(2)}$.

*Backward Constraining (re-clustering phase 1)*

Given the set of feasible vectors $\tilde{\mathbf{x}}^k_1$, $k = 1, \ldots, Q_1$, which are found in Step 3 of Algorithm 2, it is possible to run again the clustering algorithm and find, in a similar way as done in step 3 of Algorithm 1, a new set of $\bar{P}_1$ bounding boxes $\bar{\mathcal{B}}^{(1)}$, $j = 1, \ldots, \bar{P}_1$, for phase 1. This usually results in the shrinking of the previously found set of boxes for phase 1, and possibly in the elimination of some of them. Denote the new set of bounding boxes for phase 1 as $\bar{\mathcal{B}}^{(1)}$, which represents the

final pruned search space for phase 1. Note that this step defers from the backward constraining step in the original GASP algorithm, as it is based on re-clustering the previous phase, and not on the backward propagation of time constraints.

*Algorithm 3: Global Optimisation of the Pruned Search Space*

Even after pruning it is still highly likely given the nature of the search space that many local minima are still present. Hence global optimisation is an important tool needed to produce a solution to the problem.

The output from the pruning algorithm has been specifically formulated such that any global optimiser can be used to find a solution. One of the key features of this pruning algorithm is its ability to locate solution *families*. As the time of arrival at the first phase is equal to the time of departure for the second phase, it is possible to associate each of the bounding boxes found in the first phase with one or more boxes found in the second phase. Suppose that box $\bar{B}_k^{(1)}$ from the first phase is associated with box $B_l^{(2)}$ from the second phase to form a solution family with index $s$. Denote the search space associated with solution family $s$ as $B_s = \bar{B}_k^{(1)} \times B_l^{(2)}$. Assume that $S$ solution families are identified. Identifying solution families is an important feature as it prevents impossible combinations of bounding boxes from being defined.

In the case of this example mission with two deep space manoeuvres the decision variable is 11-dimensional:

$$\begin{aligned}\mathbf{x} = [&t_0,\, T_{of}^{(1)}, r^{(1)}, \alpha^{(1)}, \theta^{(1)}, \phi^{(1)} \cdots \\ &T_{\text{of}}^{(2)}, r^{(2)}, \alpha^{(2)}, \theta^{(2)}, \phi^{(2)}]^T\end{aligned} \tag{3.19}$$

1. Define two positive integers $N_4$ and $N_5$, where $N_5 >> N_4$, and $N_4 > \dim(\mathbf{x})$. For each solution family $s = 1, ...S$, use a global optimiser to solve the following problem allowing $N_4$ iterations:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \Delta v_{\text{dep}}^{(1)} + \Delta v_{\text{DSM}}^{(1)} + \Delta v_{\text{ga}}^{(1)} + \Delta v_{\text{DSM}}^{(2)} + \Delta v_{\text{b}}^{(2)} \tag{3.20}$$

subject to

$$\begin{aligned}\mathbf{x} &\in B_s \\ \begin{bmatrix} \Delta v_{\text{dep}}^{(1)} \\ \Delta v_{\text{DSM}}^{(1)} \\ \mathbf{v}_{\text{in}}^{(1)} \\ \mathbf{v}_{\text{out}}^{(2)} \\ \Delta v_{\text{DSM}}^{(2)} \\ \Delta v_{\text{b}}^{(2)} \end{bmatrix} &= \mathbf{h}(\mathbf{x}) \\ \Delta v_{\text{ga}}^{(1)} &= q_1(\mathbf{v}_{\text{in}}^{(1)}, \mathbf{v}_{\text{out}}^{(2,i)}, r_{\text{min}}^{(1)})\end{aligned} \tag{3.21}$$

Notice that the bounding boxes found by the pruning method approximate the feasible regions with respect to the inequality constraints associated with the original optimisation problem (mainly constraints on the various $\Delta v$ magnitudes). Because of this, it is possible to ignore such constraints at the final

optimisation step, and simply check the solutions found for feasibility with respect to such constraints. This is what we have done in the case studies presented below. Alternatively, the inequality constraints could be considered by the global optimisation algorithm using methods such as those described in [8].

This results in $S$ (sub-optimal) decision vectors $\mathbf{x}_k, k = 1, \ldots, S$, one for each solution family. Each of these solution vectors contains useful information to the mission analyst, since each of them corresponds to the best feasible solution (with $N_4$ iterations) found for the corresponding solution family (notice that each solution family is associated with a feasible launch window), and hence each solution gives an upper bound for the objective function within each solution family. Out of these solution vectors, the one that gives the lowest value of the objective function is denoted as the best solution found with $N_4$ iterations. Denote as $s^*$ the index of the solution family corresponding to the best solution found to $N_4$ iterations.

2. For solution family $s^*$ selected in step 1, use a global optimiser to solve the optimisation problem defined by Equations (3.20) and (3.21) allowing $N_5$ iterations of the global optimiser. Store the best value of the decision vector $\mathbf{x}^*$ found in this step, and the corresponding objective function value.

   The purpose of step 1 is to do a brief evaluation of the solution families to select the one which is most likely to contain the best solution, based on the progress of the global optimiser after $N_4$ iterations. Step 2 then optimises the solution family selected in step 1 to a greater number of iterations. It is assumed that all other tuning parameters of the optimisation algorithm are the same in steps 1 and 2.

### 3.2.6 Pruning algorithm for more than two phases

Generalising the pruning algorithm described in the previous section to the more general case of $n$ phases is not difficult. The following points should be considered.

- *Launch phase:* the pruning is performed by using Algorithm 1 as described above.
- *Intermediate phases:* These are dealt with by using Algorithm 2 as described above, but without calculating a braking manoeuvre, followed by the application of the backward constraining step. Notice that when Algorithm 2 is applied at phase $k$, the subsequent backward constraining step is carried out on phase $k - 1$.
- *Final phase:* this is pruned by using Algorithm 2 as described above, followed by the application of the backward constraining step.

### 3.2.7 Complexity Analysis

This section will analyze the complexity of the pruning algorithm. It will be shown that the time complexity algorithm scales linearly as the number of phases in a mission increases.

An important subroutine called by the trajectory pruning algorithm is the Lambert Solver. Evaluating one phase, the Lambert solver is called at least once per objective function evaluation when there are no deep space manoeuvres, and multiple times when manoeuvres are present. In general if there are $n$ deep space manoeuvres in a phase, then there will be $n+1$ calls to the Lambert Solver. Since the objective function of the mission is not considered by the proposed pruning method, and the objective function employed for pruning varies between phases, is is not possible to measure the complexity of the pruning algorithm in terms of the number of evaluations of a single objective function. For this reason the following complexity analysis is based on the number of calls to the Lambert solver.

### 3.2.8 Pruning Complexity

As the algorithm does not require the search space to be discretised, the parameters which define a mission phase can be within any user given bounds and the analysis will still hold. The algorithm can be broken down into three distinct parts:

1. Launch phase
2. Subsequent phases
3. Swingby application

The notation described in table 3.1 is used for this analysis.

*Launch Phase*

This analysis corresponds to Algorithm 1, step 2. We consider the launch phase separately to other phases for two reasons. According to Algorithm 1, the local optimiser can be started from anywhere within the search space sub domain noted by $\Omega_1$. In later phases, the optimiser is restricted to windows defined by previous phases. Secondly there is an extra constraint considered here that is not considered in other stages. As a result the local optimiser is allowed to iterate further to try and find a feasible point.

If a trajectory is purely ballistic (i.e. no DSM's are present) then only one call to the Lambert solver is required per objective function evaluation. However, as the trajectory becomes more complicated, more Lambert solutions are required. The relationship between the number of deep space manoeuvres in the launch phase, and the number of calls to the Lambert solver per objective function evaluation can be expressed as follows:

$$l_1 = s_1(d_1 + 1) \tag{3.22}$$

As there will be $k_1$ evaluations of the objective function during this phase, the number of Lambert solutions for the launch phase is given by:

| | |
|---|---|
| $L_p$ | total number of calls to Lambert solver for the pruning stage |
| $L_g$ | total number of calls to Lambert solver for the global optimisation stage |
| $L_g$ | total number of calls to Lambert solver for the pruning and global optimisation stages |
| $n$ | number of phases |
| $s_1$ | number of local optimisations in the first phase |
| $s_i$ | number of local optimisations per window ($i > 1$) in subsequent phases |
| $d_i$ | number of DSM's in phase $i$ |
| $w_i$ | number of departure windows found in phase $i$ |
| $k_1$ | number of objective function evaluations in phase 1 when executing Algorithm 1, step 2 |
| $k_{2,i}$ | number of objective function evaluations in phase $i$ when executing algorithm 2 |
| $k_2$ | upper bound on the number of objective function evaluations in all phases when executing algorithm 2 |
| $k_3$ | upper bound on the number of objective function evaluations in all phases when executing algorithm 3 |

Table 3.1: Complexity analysis notation

$$l_1 = s_1(d_1 + 1)k_1 \tag{3.23}$$

Typical values for $k_1$ lie in the interval $[300, 350]$.

*Remaining Phases*

This analysis corresponds to Algorithm 2, step 2. The complexity structure is very similar for the remaining phases to that of the first phase. The number of Lambert solutions required to carry out Algorithm 2 in phase $i$ can be calculated from:

$$l_2^{(i)} = s_i w_i (d_i + 1)k_{2,i} \tag{3.24}$$

In equation 3.24 the term $s_i w_i$ is equal to the total number of local optimisations carried out in that stage. If we now combine all the phases of the mission equation 3.25 indicates how many calls to the Lambert solver are made as a result of applying Algorithm 2 in all phases.

$$l_2 = \sum_{i=2}^{n} l_2^{(i)} = \sum_{i=2}^{n} s_i w_i (d_i + 1)k_{2,i} \tag{3.25}$$

If we consider an upper bound on the number of function evaluations per phase when applying step 2 of Algorithm 2, such that $k_2 >= k_{2,i}$, $i = 2, \ldots n$, then is is possible to find an upper bound on the number of Lambert evaluations as a result of applying step 2 of Algorithm 2 in all phases:

$$l_2 < k_2 \sum_{i=2}^{n} s_i w_i (d_i + 1) \tag{3.26}$$

*Swingby application*

This analysis corresponds to Algorithm 2, step 3. Applying the swing by manoeuvre to link consecutive phases requires more effort from the local optimiser. This is represented by increasing the maximum number of iterations the optimiser can run for. In this case, the number of objective function evaluations is in most practical cases much larger than $k_1$ and $k_2$. Consider an upper bound on the number of function evaluations per phase when applying step 3 of Algorithm 2, $k_3$. Then the total number of Lambert solver calls when applying step 3 of Algorithm 2 obeys the following inequality:

$$l_3 < k_3 \sum_{i=2}^{n-1} s_i w_i (d_i + 1) \tag{3.27}$$

*Total complexity of pruning stage*

Combining the launch, deep space flight and swingby complexity equations given by 3.23, 3.26 and 3.27 respectively produces the following inequality, which is a measure of the total pruning complexity with respect to the number of Lambert solutions required:

$$L_p < s_1(d_1 + 1)k_1 + k_2 \sum_{i=2}^{n} \{s_i w_i (d_i + 1)\} + k_3 \sum_{i=2}^{n-1} \{s_i w_i (d_i + 1)\} \tag{3.28}$$

Assume that $s_i \leq \bar{s}$, $i = 1, \ldots, n$, and the number of deep space manoeuvres in each phase is bounded by $d_i \leq \bar{d}$, $i = 1 \ldots n$. Let $\gamma = \bar{s}(\bar{d} + 1)$. Then inequality 3.28 can now be rewritten as:

$$L_p < \gamma k_1 + k_2 \sum_{i=2}^{n} (w_i \gamma) + k_3 \sum_{i=2}^{n-1} (w_i \gamma) \tag{3.29}$$

Assume that $w_i$ is bounded for each phase after launch, so that $w_i < \bar{w}$, $i = 2, \ldots, n$, then for any given mission with $n$ phases the total complexity (measured as an upper bound on the number of Lambert solver calls required for pruning the search space) can be described by

$$L_p < \gamma k_1 + k_2 (n - 1)\bar{w}\gamma + k_3 (n - 2)\bar{w}\gamma \tag{3.30}$$

Denote $\Psi = \bar{w}\gamma$ and rewrite inequality (3.30) as follows:

$$L_p < \gamma k_1 + k_2 (n - 1)\Psi + k_3 (n - 2)\Psi \tag{3.31}$$

or

$$L_p < \gamma k_1 - k_2 \Psi - 2k_3 \Psi + (k_2 + k_3)\Psi n \tag{3.32}$$

Looking at inequality (3.32), it can clearly be seen that the relationship between the number of phases in a mission ($n$) and the upper bound on the number of

Lambert solutions required is linear. This implies that the algorithm scales well as more phases are added. The three constants $k_1$, $k_2$ and $k_3$ can be estimated from experience. For a typical mission with 1 deep space manoeuvre in each phase, typical values for $k_1$, $k_2$ and $k_3$ are 320, 250, and 550, respectively.

**Complexity of the global optimisation stage**

Note that the above analysis is only for the pruning stage, so that it excludes the subsequent global optimisation stage. The complexity of the global optimisation stage can be defined as follows. Evaluating the whole trajectory with the purpose of calculating the objective function requires a number of Lambert evaluations which is upper bounded as follows:

$$L_f = \sum_{i=1}^{n}(d_i + 1) < (\bar{d} + 1)n \tag{3.33}$$

If a population based algorithm with $N_p$ individuals is employed over $N_i$ generations over each solution family located, and the most promising solution family is optimised by the same algorithm over $N_g$ generations, then the total number of Lambert calls for the global optimisation stage is upper bounded as follows:

$$L_g < N_p N_i S(\bar{d} + 1)n + N_p N_g(\bar{d} + 1)n = (N_p N_i S + N_p N_g)(\bar{d} + 1)n \tag{3.34}$$

where $S$ is the number of solution families located.

**Complexity of pruning and global optimisation**

The total complexity of the pruning stage followed by the global optimisation stage, in terms of the number of Lambert solver calls, is upper bounded as follows

$$L < \gamma k_1 - k_2\Psi - 2k_3\Psi + (k_2 + k_3)\Psi n + (N_p N_i S + N_p N_g)(\bar{d} + 1)n \tag{3.35}$$

Again, it is clear that the complexity grows linearly with the number of phases when considering both the pruning and the global optimisation stages.

### 3.2.9 Results

In his section it will be shown how effective this pruning algorithm can be. Three missions will be described and then pruned and optimised. A grid sampling method will be compared on a simple mission to illustrate the advantages of pruning.

**Earth-Mars (grid sampling comparison)**

To illustrate the advantages gained by applying the pruning algorithm a simple one phase mission from Earth to Mars is examined. The mission consists of a transfer with a deep space manoeuvre and an insertion manoeuvre at Mars. The mission can

be characterised by the following 6 decision variables, each of which has an initial feasible interval:

- $t_0 \in [2000, 3000]$ MJD2000
- $T_{\text{of}} \in [150, 450]$ days
- $r \in [1.496e8, 2.2794e8]$ km
- $\theta \in [-\pi/6, \pi/6]$ rad
- $\phi \in [-\pi/6, \pi/6]$ rad
- $\alpha \in [0.1, 0.9]$

An insertion manoeuvre at Mars is specified with radius of pericenter $r_p = 3950$ km, and eccentricity $e = 0.98$. The impulsive manoeuvres were constrained as follows:

- $\Delta v_{\text{dep}} \leq 5\,\text{km/s}$
- $\Delta v_{\text{DSM}} \leq 2\,\text{km/s}$
- $\Delta v_{\text{b}} \leq 3\,\text{km/s}$

For the grid sampling, 45 points were taken along the $t_0$ interval, 20 along the $T_{\text{of}}$ interval, 20 along the $r$ interval, 10 along the $\theta$ interval, 10 along the $\phi$ interval, and 10 along the $\alpha$ interval. This gives a total of 18 million points to be sampled, of which only 19 points were found to be feasible. The grid sampling required 36 million calls to the Lambert solver, and took almost 3.36 hours on an Intel Core 2 Duo 2.0 GHz PC running Matlab 2007a.

The sequential quadratic programming algorithm, as implemented in function `fmincon` of Matlab's optimisation Toolbox was employed for the local optimisation steps associated with the proposed pruning method. To perform the pruning, 150 random vectors were generated in phase 1 as described in section 3.2.2 ($N_1 = 150$), and the local optimiser was started from each vector, resulting in 89 feasible vectors ($M_1 = 89$). The mean shift clustering algorithm was run with a bandwidth value of 230 to find the approximate feasible regions. A total of 116,326 calls to the Lambert solver where done by the proposed pruning algorithm in this case, and the pruning took 128 seconds on the same PC. This is only 1.06% of the time spent by the same PC to perform the grid sampling. The clustering algorithm takes a negligible amount of time to execute compared with the overall time it takes to perform the pruning with the proposed method.

Figure 3.2(a) shows the projected bounding boxes found using the proposed pruning method, while figure 3.2(b) shows the projected points found using the grid sampling method. Notice that the feasible points obtained by means of grid sampling are located inside the bounding boxes found by the proposed pruning method. This has been verified using the numerical values obtained.

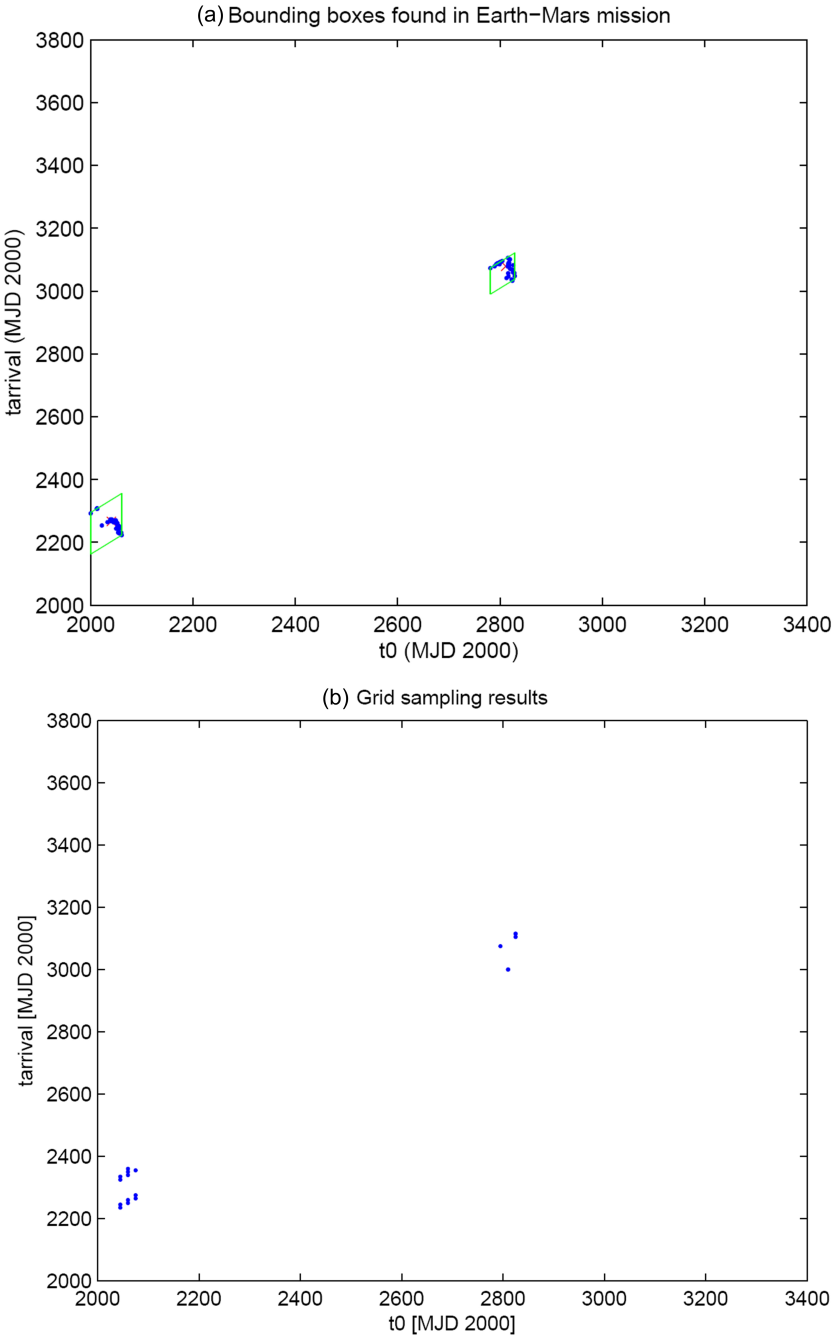



Figure 3.2: (a) Illustration on the located bounding boxes on the $t_0 - t_{\text{arr}}$ plane for the estimates of the feasible regions for the Earth-Mars mission The actual bounding boxes are in six dimensions. (b) Projection on the $t_0 - t_{\text{arr}}$ plane of the feasible points located by grid sampling in the case of the Earth-Mars mission. The actual points are in six dimensions

Notice the difference in computations between grid sampling and the proposed pruning method, given the amount of feasible vectors found by each method. Increasing the sampling resolution to enable the grid sampling method to find more feasible points would result in even heavier computations.

**Earth-Venus-Mars**

To further test the proposed pruning method, we have defined an Earth-Venus-Mars mission with one deep space manoeuvre between Venus and Mars. The problem has 7 decision variables. The initial ranges for all variables are defined below:

- $t_0 \in [36507302]$ MJD2000
- $T_{\text{of}}^{(1)} \in [50, 400]$
- $T_{\text{of}}^{(2)} \in [50, 700]$
- $r \in [1.0821e8, 2.2794e8]$
- $\theta \in [-\pi, \pi]$
- $\phi \in [-\pi/8, \pi/8]$
- $\alpha \in [0.1, 0.9]$

All impulsive manoeuvres were constrained as follows:

- $\Delta v_{\text{dep}} \leq 5\,\text{km/s}$
- $\Delta v_{\text{ga}} \leq 5\,\text{km/s}$
- $\Delta v_{\text{DSM}} \leq 2\,\text{km/s}$
- $\Delta v_{\text{b}} \leq 3\,\text{km/s}$

An insertion manoeuvre at Mars is specified with radius of pericenter $r_p = 3950$ km, and eccentricity $e = 0.98$.

Figures 3.3(a) and 3.3(b) show the projected bounding boxes for each phase. These diagrams illustrate how an individual box in the first phase can be related to an individual box in the second phase to form a solution family. Six solution families can be identified.

To perform the pruning for each phase, 120 random points were generated in phase 1 as described in section V ($N_1 = 120$), and the local optimiser was started from each point. Similarly, 350 random initial points were generated in phase 2 ($N_2 = 350$, 50 points per feasible $t_{\text{dep}}^{(2)}$ interval, with seven initial feasible intervals $\mathcal{I}_1, \ldots, \mathcal{I}_6$). Four local optimisations from each initial feasible point in phase 2 were performed ($N_3 = 4$), to evaluate feasibility with respect to the gravity assist manoeuvre (Step 3 of Algorithm 2). After the gravity assist calculations and backward constraining, 220 feasible vectors were found in phase 1 ($Q_1 = 220$), while 65 vectors

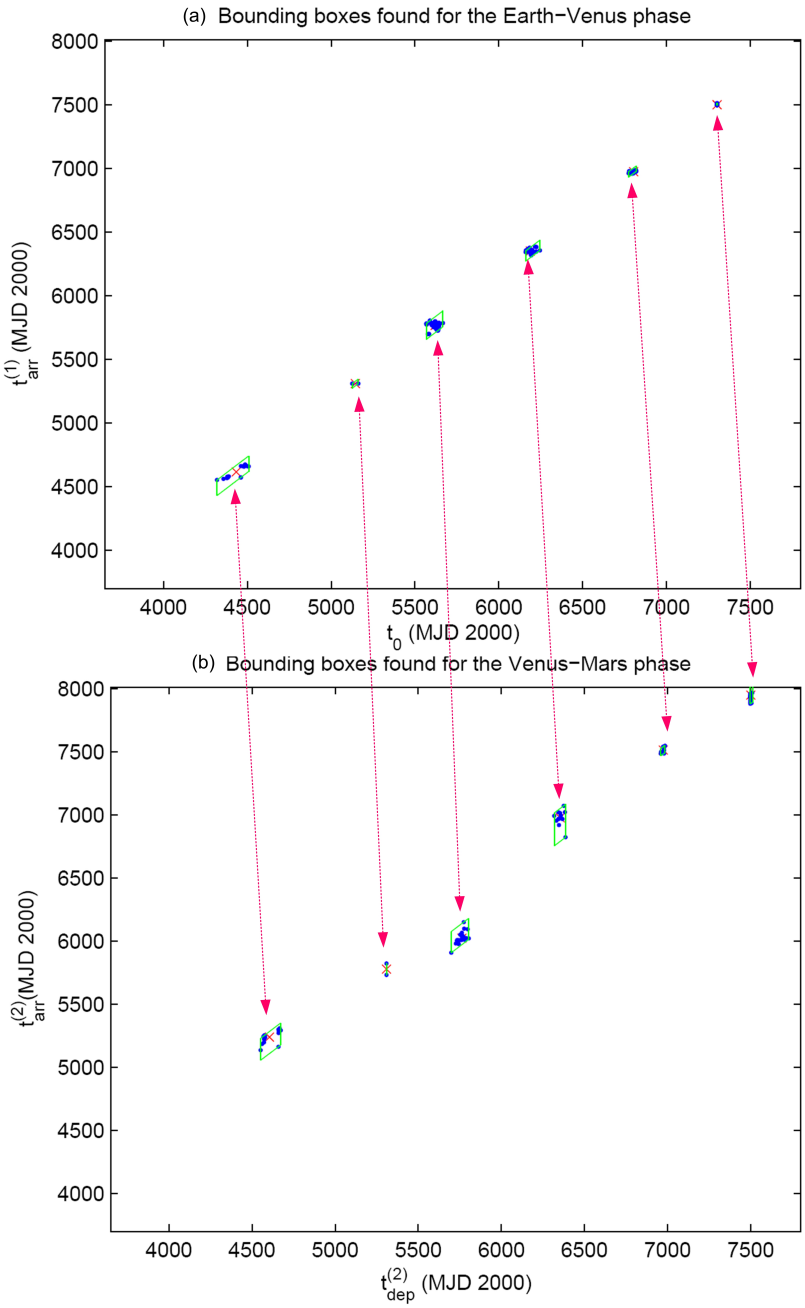



Figure 3.3: (a) Illustration on the located bounding boxes on the $t_0 - t_{\text{arr}}^{(1)}$ plane for the estimates of the feasible regions for the Earth-Venus phase. The actual bounding boxes are in six dimensions. (b) Projection on the $t_{\text{dep}}^{(2)} - t_{\text{arr}}^{(2)}$ plane of the located bounding boxes for the estimates of the feasible regions for the Venus-Mars phase. The actual bounding boxes are in six dimensions. The double headed arrows show the six solution families that were identified.

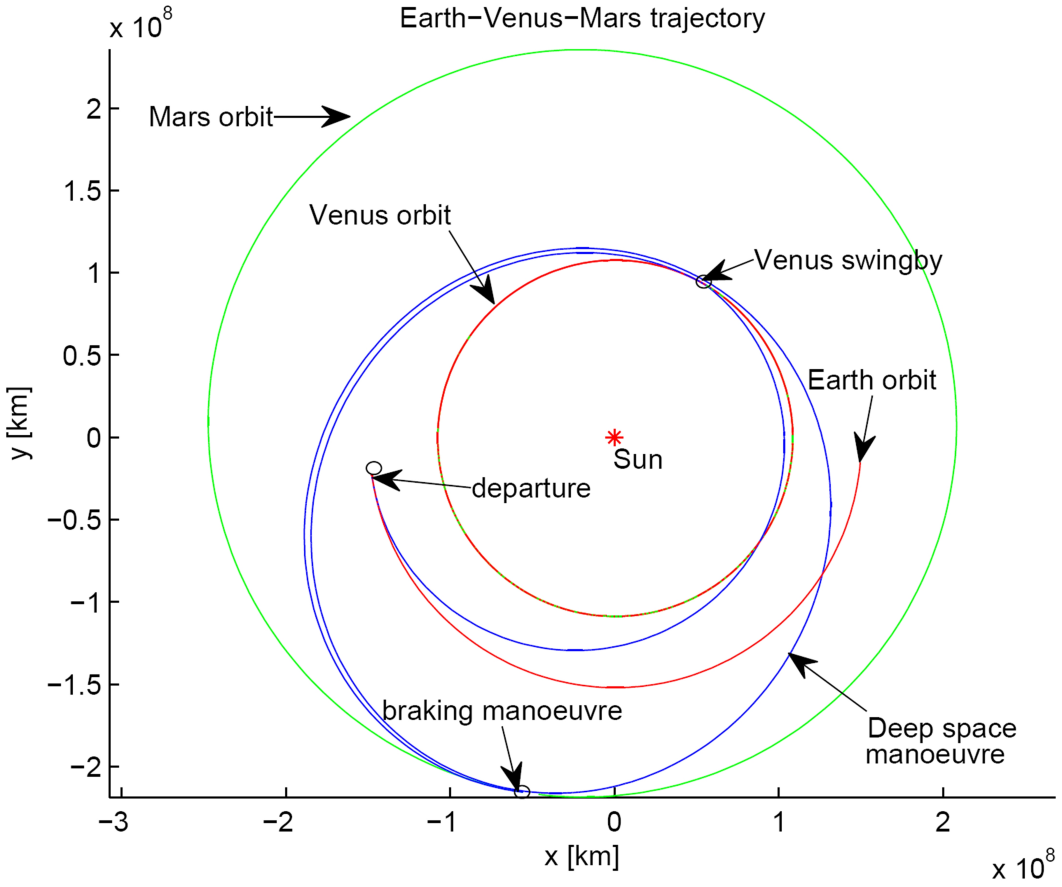



Figure 3.4: Representation of the best trajectory found when Differential Evolution was applied to the pruned search space

were left in phase 2 ($Q_2 = 65$). The mean shift clustering algorithm was run with bandwidth value of 230 and 270 in phases 1 and 2, respectively. A total of 389,805 calls to the Lambert solver were required for the pruning phase (650 s CPU time)

Differential evolution with a population size of 20 individuals and parameter values $F = 0.8$ and $CR = 0.8$ was employed to search for an optimal solution for each of the six solution families located. The algorithm was initially run for $N_4 = 200$ iterations for each solution family. This was done to select the most promising solution family. Following this, Differential Evolution was run for $N_5 = 2000$ iterations for the selected solution family. A total of 192,000 Lambert solver calls were performed during the optimisation phase (215 s CPU time). The returned result gave the following values for the decision variables: $t_0 = 4469.9$, $t_{\text{of}}^{(1)} = 171.7855$, $t_{\text{of}}^{(2)} = 682.4994$, $r^{(2)} = 1.7252 \times 10^8$, $\theta^{(2)} = -1.9133$ rad, $\phi^{(2)} = -0.0073$ rad, $\alpha^{(2)} = 0.5037$. The resulting impulsive manoeuvres were: $\Delta v_{\text{dep}}^{(1)} = 2.9743$ km/s, $\Delta v_{\text{ga}}^{(1)} = 8.547 \times 10^{-5}$ km/s, $\Delta v_{\text{DSM}}^{(2)} = 0.4729$ km/s, $\Delta v_b^{(2)} = 2.0158$ km/s, giving a total $\Delta v$ value of 5.4630 km/s. Figure 3.4 shows the corresponding spacecraft trajectory projected on the plane defined by the Earth's rotation.

Notice that the tolerance of the Lambert solver was relaxed at $10^{-6}$ for the pruning phase, and tightened at $10^{-14}$ for the Differential Evolution optimisation phase. This was done in order to save computation time, as only an estimate of the feasible regions is found through the pruning method, and the regions found are not very sensitive to the tolerance value.

### MESSENGER Mission

The MESSENGER spacecraft is set to become the first spacecraft to successfully orbit Mercury, and will send back the first new pieces of data since the Mariner 10 mission. It was launched from Earth in August 2004. It is due to complete its mission to Mercury in March 2011. On its way to Mercury it will have completed one flyby of Earth, two flyby's of Venus and 3 flyby's of Mercury. This mission is of particular interest because of the complexity involved. The mission contains:

1. A launch
2. 6 powered swingbys
3. 7 deep space manoeuvres
4. An orbital insertion manoeuvre

In order to try and replicate the actual mission flown, the model needed to be updated. The new model allowed for Lambert problem solutions to include multiple revolutions of a body. In addition to this, both high and low energy transfers can take place when the number of complete revolutions is $> 1$. In the test case being presented here, the integer variables representing the number of revolutions is kept constant and not optimised. The same is true for the binary variables defining a high or low energy transfer.

The MESSENGER mission involves a decision variable in 36 dimensions with a total of 15 $\Delta V$'s. The insertion manoeuvre that occurs after the third swingby of Mercury is defined with a radius of pericenter $r_p = 2640$ km, and eccentricity $e = 0.7396$. The constraints imposed on the impulsive manoeuvres are given below:

- $\Delta v_{\text{dep}} \leq 5\,\text{km/s}$
- $\Delta v_{\text{ga}}^{(1..6)} \leq 2\,\text{km/s}$
- $\Delta v_{\text{DSM}}^{(1..7)} \leq 3\,\text{km/s}$
- $\Delta v_{\text{b}} \leq 4\,\text{km/s}$

The bounds used for this mission are:

- $t_0 \in [1000, 4000]$
- $T_{\text{of}}^{(1..7)} \in [200, 500]$
- $r^{(1)} \in [1.3464e8, 1.6456e8]$
- $r^{(2)} \in [1.0821e8, 1.4960e8]$
- $r^{(3)} \in [9.739e7, 1.1903e8]$
- $r^{(4)} \in [5.791e7, 1.0821e8]$
- $r^{(6..7)} \in [5.212e7, 6.370e7]$
- $\theta^{(1..7)} \in [-\pi, \pi]$
- $\phi^{(1..7)} \in [-\pi/6, \pi/6]$
- $\alpha^{(1..7)} \in [0.1, 0.9]$

For this mission the pruning algorithm was used with the following parameters. Number of randomly generated starts of the local optimiser in the first phase, $N_1 = 150$. Number of randomly generated starts per window in subsequent phases $N_i = 80$ where $i > 1$. The maximum number of iterations the local optimiser is allowed to run for when a swingby is being considered was 450 and 150 when there was no swingby included. Each feasible point found before a swingby manoeuvre was allowed a maximum of 3 attempts to find a matching feasible point after the swingby is included. The best solution found is described in table 3.2

The solution vector produces the impulsive manoeuvres listed in table 3.3. Looking at the results, there is only one constraint violation in the solution. The violation occurs at the deep space manoeuvre between Venus and Mercury. The size of the violation is notably small. The 14 remaining impulsive manoeuvres all satisfy the constraints with ease.

The bounding boxes generated during the first and last pruning phases of the MESSENGER mission are shown in figures 3.5 and 3.6. The trajectory generated from the solution vector shown in table 3.2 that produces the results illustrated in table 3.3 is shown graphically in figure 3.7.

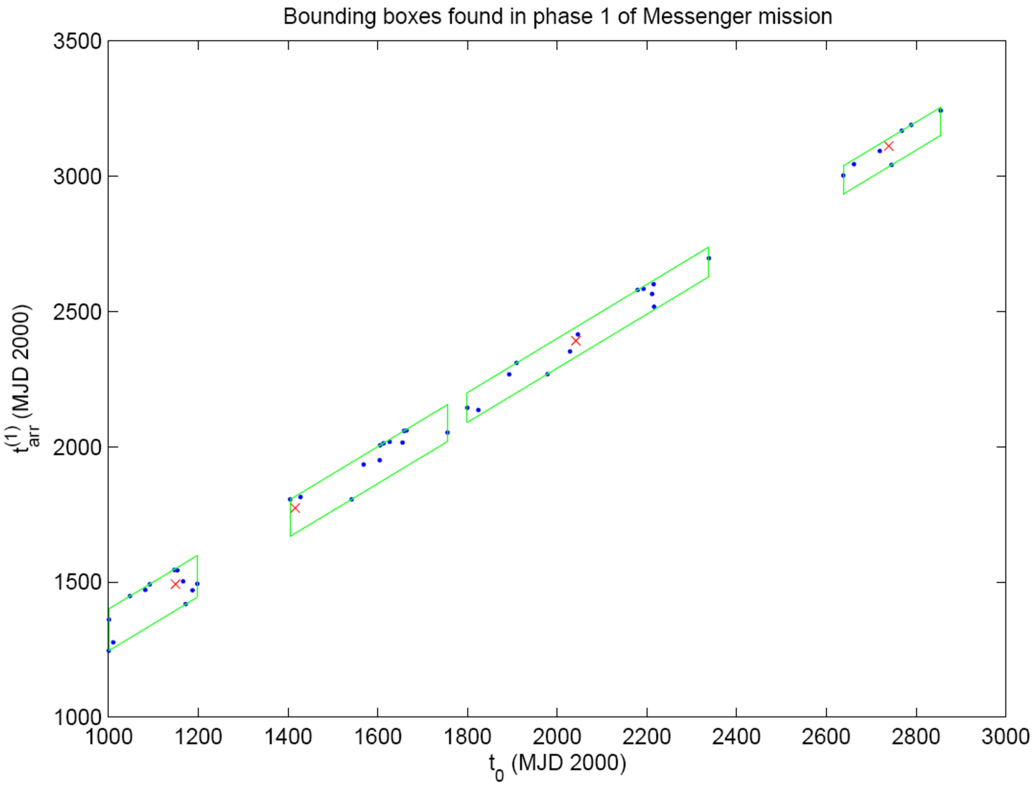



Figure 3.5: Projection on the $t_0 - t_{\text{arr}}^{(1)}$ plane of the located bounding boxes for the estimates of the feasible regions for phase 1 of the MESSENGER mission. The actual bounding boxes are in six dimensions

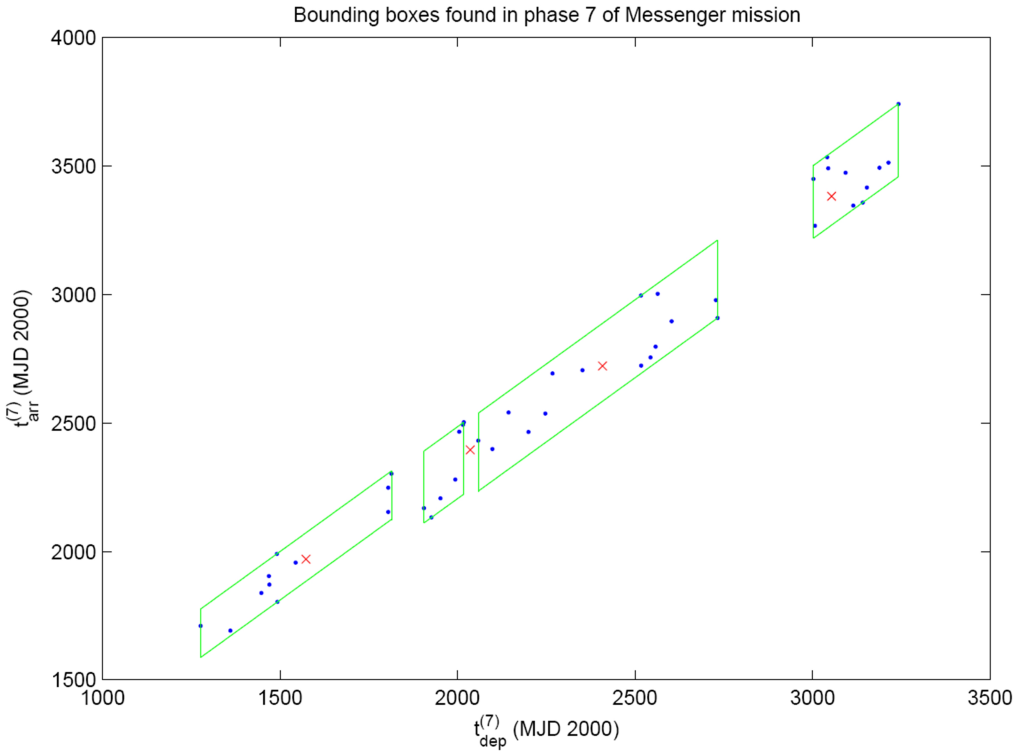



Figure 3.6: Projection on the $t_{\text{dep}}^{(7)} - t_{\text{arr}}^{(7)}$ plane of the located bounding boxes for the estimates of the feasible regions for phase 7 of the MESSENGER mission. The actual bounding boxes are in six dimensions

| index | $t$ (MJD2000) | $T_{\text{of}}$ (days) | $r$ (km) | $\theta$ rad | $\phi$ rad | $\alpha$ |
|---|---|---|---|---|---|---|
| 0 | 1527.3 | | | | | |
| 1 | | 399.9972 | 1.4993e+8 | -3.1117 | 6.0655e-6 | 0.4658 |
| 2 | | 377.1584 | 1.3667e+8 | -1.4910 | 0.0176 | 0.6430 |
| 3 | | 177.7119 | 1.0398e+8 | 2.2906 | 6.7708e-4 | 0.3966 |
| 4 | | 171.7616 | 5.8483e+7 | -2.8456 | 0.0197 | 0.4707 |
| 5 | | 126.5759 | 6.7573e+7 | -1.1223 | 3.6121e-4 | 0.5166 |
| 6 | | 132.5413 | 5.2811e+7 | -2.2575 | -0.0053 | 0.4979 |
| 7 | | 107.1364 | 6.7171e+7 | -1.0448 | -6.0022e-5 | 0.6219 |

Table 3.2: Decision vector values found after optimizing the pruned MESSENGER search space.

### BepiColombo Mission

The BepiColombo mission is another mission that utilizes the inner planets in order to reach Mercury. This mission has been slightly simplified in order to work with the model we are using. The actual mission includes a low thrust arc transfer which cannot be replicated with this model. Although the mission studied here is inspired by BepiColombo, the results obtained should not be compared with the original BepiColombo because of the difference in the sequence. The sequence has also been reduced to Earth-Earth-Venus-Venus-Mercury. There is a deep space manoeuvre in each phase giving a 21 dimensional search space. The bounds used for this mission are:

- $t_0 \in [4000, 8000]$
- $T_{\text{of}}^{(1..4)} \in [100, 800]$
- $r^{(1)} \in [1.3464e8, 1.6456e8]$
- $r^{(2)} \in [1.0821e8, 1.4960e8]$
- $r^{(3)} \in [9.7388e7, 1.1903e8]$
- $r^{(4)} \in [5.7909e7, 1.0821e8]$
- $\theta^{(1..4)} \in [-\pi, \pi]$
- $\phi^{(1..4)} \in [-\pi/6, \pi/6]$
- $\alpha^{(1..4)} \in [0.1, 0.9]$

Subject to the following constraints:

- $\Delta v_{\text{dep}} \leq 3\,\text{km/s}$
- $\Delta v_{\text{ga}}^{(1..3)} \leq 3\,\text{km/s}$
- $\Delta v_{\text{DSM}}^{(1..4)} \leq 2\,\text{km/s}$

| Manoeuvre | Type | Bodies Involved | $\Delta V(km/s)$ | Constraint Violation |
|---|---|---|---|---|
| 1 | Launch | E | 0.127137 | No |
| 2 | DSM | E-E | 0.074635 | No |
| 3 | DSM | E-V | 2.253777 | No |
| 4 | DSM | V-V | 1.621058 | No |
| 5 | DSM | V-Me | 3.085997 | Yes |
| 6 | DSM | Me-Me | 0.094500 | No |
| 7 | DSM | Me-Me | 0.166976 | No |
| 8 | DSM | Me-Me | 0.063729 | No |
| 9 | Swingby | E | 0.509242 | No |
| 10 | Swingby | V | 0.005469 | No |
| 11 | Swingby | V | 0.843764 | No |
| 12 | Swingby | Me | 1.774886 | No |
| 13 | Swingby | Me | 0.015440 | No |
| 14 | Swingby | Me | 0.001736 | No |
| 15 | Insertion | V | 0.275635 | No |
| | | Total $\Delta V$ | 10.913980 | |

Table 3.3: MESSENGER mission's deep space manoeuvres resulting from the best solution vector

- $\Delta v_{\mathrm{b}} \leq 3\,\mathrm{km/s}$

The bounding boxes generated during the first and last pruning phases of the BepiColombo mission are shown in figures 3.8 and 3.9.

The solution vector is shown below in table 3.4. This solution found gives a total $\Delta V$ of 1.195605. A break down of all the impulsive manoeuvres is shown in table 3.5. Figure 3.10 shows the trajectory graphically.

Ideally there should be an insertion manoeuvre at Mercury. There are reasons for not including such a manoeuvre in this test case. The full BepiColombo mission contains two further orbits of Mercury with a deep space manoeuvre in each phase. As the actual mission contained extra phases, it is unknown weather a suitable parking orbit can be achieved when arriving directly from Venus. The purpose of studying this mission was to show that the search space could be pruned and a good solution found, which we have done. Secondly the Earth-Venus-Mars case and the MESSENGER mission presented above have demonstrated that we can prune and optimise the search space when such insertions are required.

## 3.3 Optimal Sequence Selection

### 3.3.1 Introduction

A pruning algorithm that is able to deal with any specified sequence of planets has been developed and described in Section 3.2. This section describes how the sequence of planets can be chosen and optimised when a departure and arrival planet are

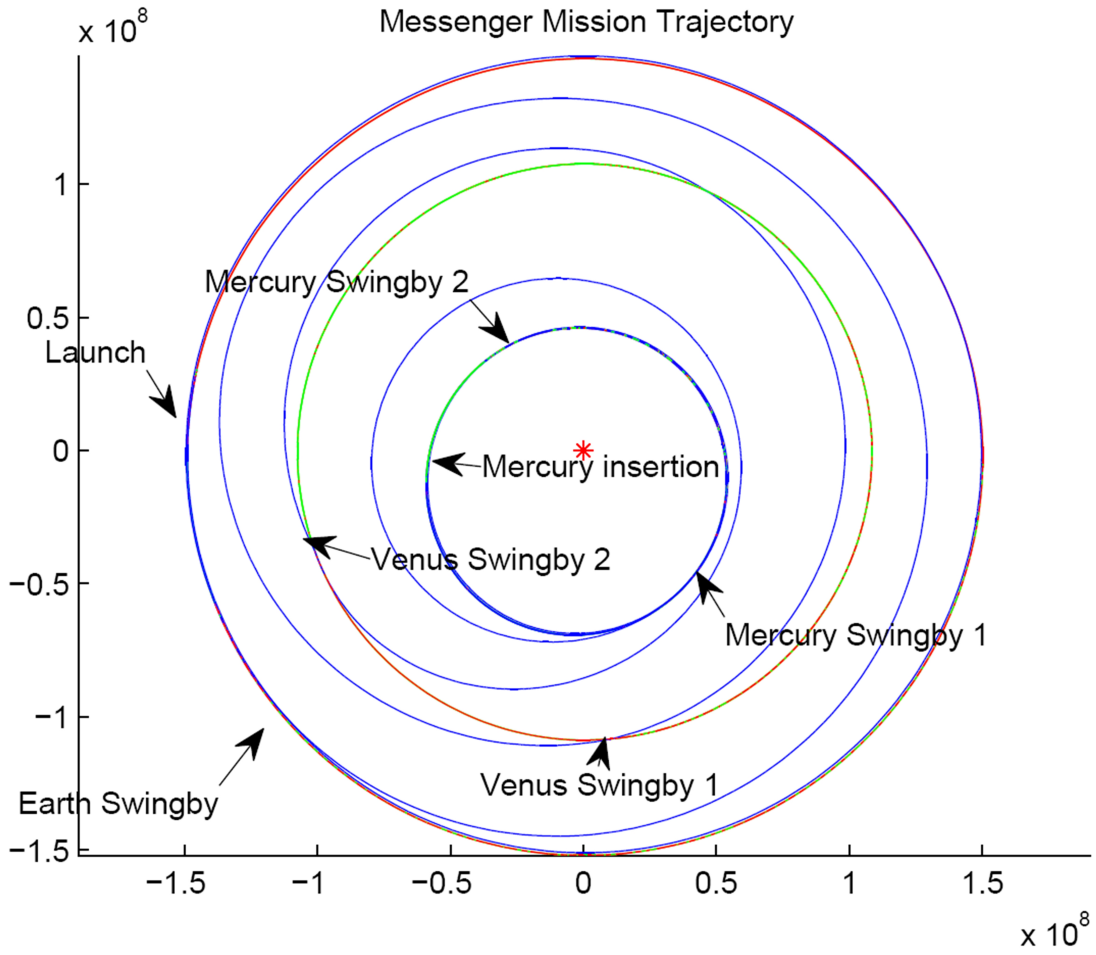



Figure 3.7: Optimised MESSENGER Trajectory

chosen. The algorithm also determines how many deep space manoeuvres (if any) should be placed in each phase of the mission. A method for inserting retrograde transfers and multiple revolutions of bodies is also introduced.

The problem of optimizing planetary sequences can be formulated as an integer optimisation problem. However determining an optimal sequence requires a trajectory to be created and compared to other trajectories from alternate sequences. The algorithm developed uses integer optimisation techniques to generate candidate sequences. The objective function used is the sum of $\Delta V$'s generated from the pruning algorithm described in the previous section. This method is henceforth termed as the *hybrid* algorithm.

In the next section Differential Evolution and Particle Swarm optimisation are adapted to solve integer problems and their performances compared. Following this the hybrid algorithm is described and tested on a selection of mission design problems.

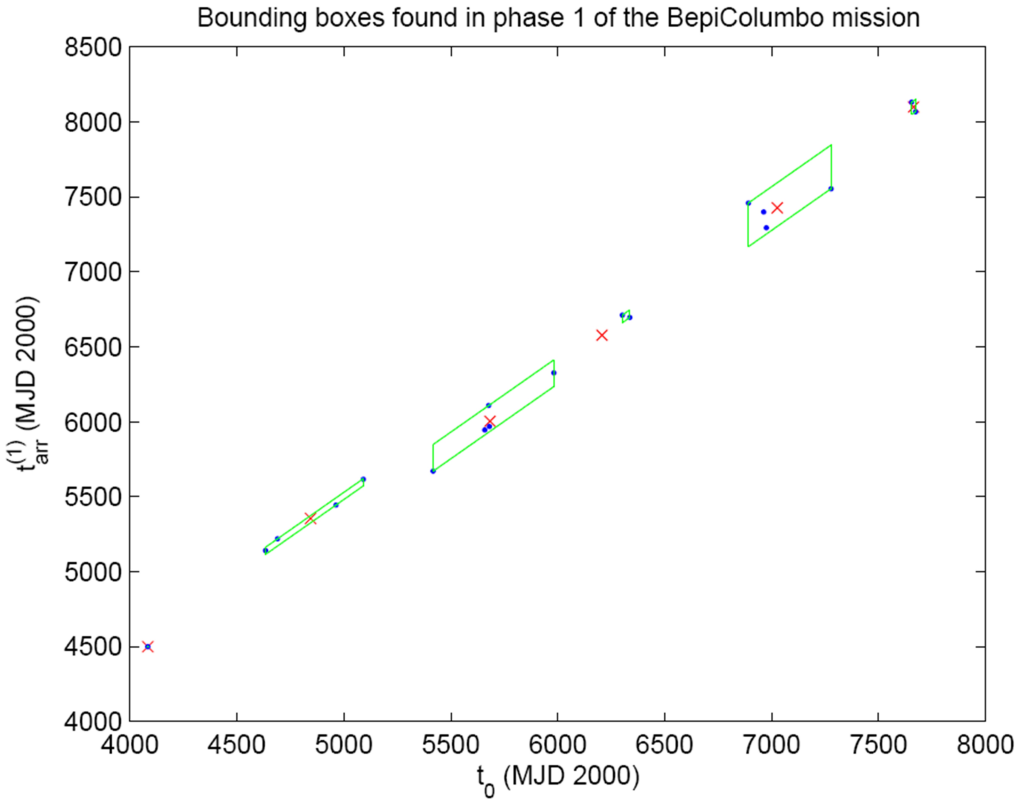



Figure 3.8: Projection on the $t_0 - t_{\text{arr}}^{(1)}$ plane of the located bounding boxes for the estimates of the feasible regions for phase 1 of the BepiColombo mission. The actual bounding boxes are in six dimensions

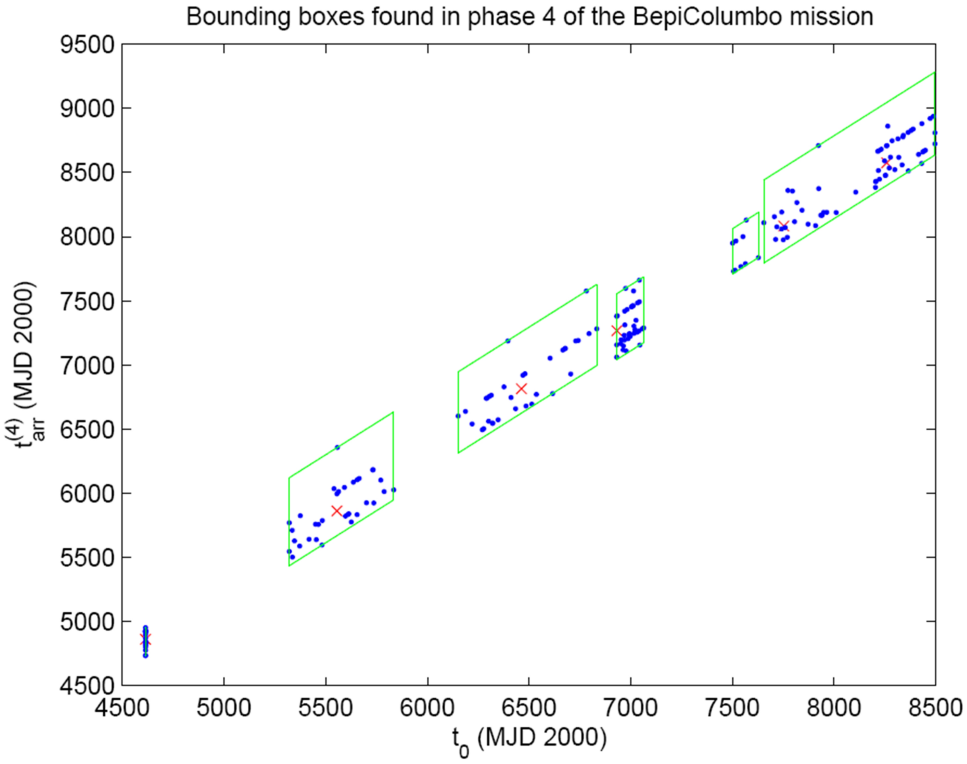



Figure 3.9: Projection on the $t^{(4)}_{\text{dep}} - t^{(4)}_{\text{arr}}$ plane of the located bounding boxes for the estimates of the feasible regions for phase 4 of the BepiColombo mission. The actual bounding boxes are in six dimensions

| index | $t$ (MJD2000) | $T_{\text{of}}$ (days) | $r$ (km) | $\theta$ rad | $\phi$ rad | $\alpha$ |
|---|---|---|---|---|---|---|
| 0 | 7.5983e+003 | | | | | |
| 1 | | 372.3406 | 1.5203e+008 | -2.8955 | 1.4951e-4 | 0.5166 |
| 2 | | 171.2483 | 1.2460e+008 | 1.4827 | 0.0143 | 0.4886 |
| 3 | | 305.9244 | 9.7390e+007 | -0.6870 | -2.6169e-5 | 0.6110 |
| 4 | | 256.0651 | 9.7822e+007 | -0.8699 | -0.0730 | 0.5178 |

Table 3.4: Decision vector values found after optimizing the pruned BepiColombo search space.

| Manoeuvre | Type | Bodies Involved | $\Delta V(km/s)$ | Constraint Violation |
|---|---|---|---|---|
| 1 | Launch | E | 0.274406 | No |
| 2 | DSM | E-E | 0.263321 | No |
| 3 | DSM | E-V | 0.010990 | No |
| 4 | DSM | V-V | 0.036779 | No |
| 5 | DSM | V-Me | 0.188297 | No |
| 6 | Swingby | E | 0.413114 | No |
| 7 | Swingby | V | 0.002458 | No |
| 8 | Swingby | V | 0.006241 | No |
| | | Total $\Delta V$ | 1.195605 | |

Table 3.5: BepiColombo mission's deep space manoeuvres resulting from the best solution vector

### 3.3.2 Non-linear Integer Programming

A simple method for adapting particle swarm optimisation for integer programming is proposed by Laskari *et al* in [9]. The method works with real decision variables, but simply rounds them off to the nearest whole integer value when they are used. The results obtained by Laskari and his team show that the rounding of the solution vector produces optimal results. We have applied the same method to Differential Evolution for comparison. A selection of problems taken from [9] were be used as a benchmark, they are listed below:

**Problem 1.** The first problem is a relatively simple polynomial minimization problem in four dimensions, where the following objective function is minimised:

$$J_1(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \tag{3.36}$$

The optimal solution is found at $x^* = [0,\ 0,\ 0,\ 0]^T$, where $J_1(x^*) = 0$.

**Problem 2.** This optimisation problem tests the ability of the optimisers to find a solution where the objective function takes on real values. The objective function has the form:

$$J_2(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2 \tag{3.37}$$

The optimal solution is found at $x^* = [0,\ 1]^T$, where $J_2(x^*) = -3833.12$

**Problem 3.** The final problem to solve is the most complex. It is a quadratic

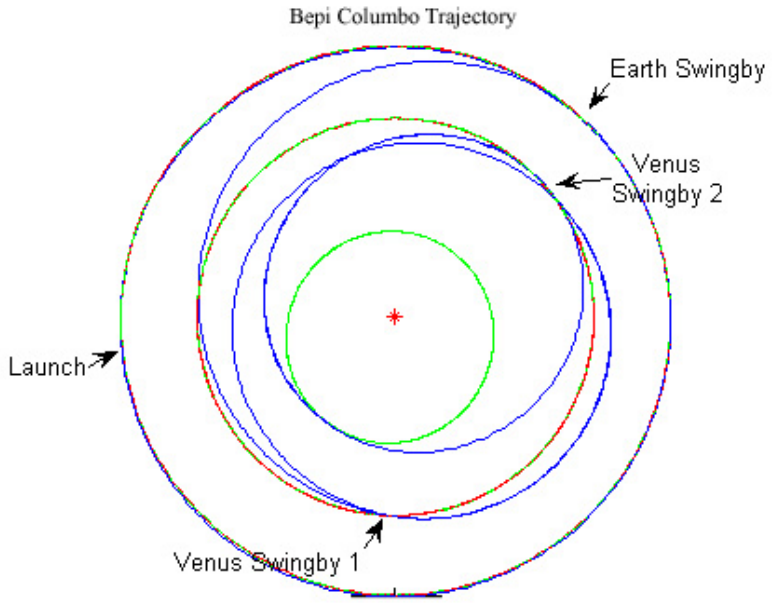



Figure 3.10: Optimised BepiColombo Trajectory

objective function in five dimensions.

$$J_3(x) = -\begin{bmatrix} 15 & 27 & 36 & 18 & 12 \end{bmatrix} x + x^T \begin{bmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -20 \\ -10 & 32 & -10 & -20 & 31 \end{bmatrix} x \quad (3.38)$$

The two best known solutions to $J_3$ are found at $x^* = [0,\ 11,\ 22,\ 16,\ 6]^T$ and $x^* = [0,\ 12,\ 23,\ 17,\ 6]^T$. Both vectors produce $J_3(x^*) = -737$.

Each problem has been optimised by Differential Evolution and then by PSO. To keep the comparisons fair the same population size was used in both algorithms. Each test was run 20 times and the results averaged. With Differential Evolution, the cross over rate $CR$ was fixed at 0.8 and the step size $F$ was set to 0.5 for all of the tests described below. For PSO, the confidence coefficients $w$ and $c$ were fixed at 0.721348 and 1.193147, respectively.

The benchmark test results are shown in table 3.6. PSO appears to outperform differential evolution in two of the test cases (Problems 1 and 3), including the most

Table 3.6: Evaluation of Differential Evolution and PSO for integer programming problems. P1-P3 correspond to Problems 1 to 3. 'Pop.' corresponds to the population size employed, 'Gen.' is the average number of generations over twenty runs to find the known solution to the problem, 'Eval.' is the average number of objective function evaluations required to find the solution.

| - | P1 DE | P1 PSO | P2 DE | P2 PSO | P3 DE | P3 PSO |
|---|---|---|---|---|---|---|
| Pop. | 20 | 20 | 20 | 20 | 70 | 20 |
| Gen. | 263 | 72 | 55 | 81 | 384 | 95 |
| Eval. | 6070 | 1400 | 930 | 1730 | 27210 | 6320 |

complex one. For this reason PSO was used as the optimiser of choice for the planet sequencing problem.

### 3.3.3 A Hybrid Approach to Planetary Sequence optimisation

#### Problem Formulation

A two level approach is used to solve the optimal planet sequencing problem. The top level is the adapted integer particle swarm optimiser. This generates a vector that can be broken down into a number of subsections. The first subsection represents each of the intermediate planets where a swingby manoeuvre will be performed. The second subsection represents the number of deep space manoeuvres to occur in that particular phase. By default the top level decision vector contains only these subsections. However, in some cases it may be necessary to add extra variables to allow for the presence of retrograde transfers or multiple revolutions of a body.

Once a sequence has been generated by the particle swarm optimiser at the top level, the corresponding objective function is evaluated the lower level by the pruning algorithm followed by a global optimisation stage, as described in Section 3.2. In order to improve the time performance of the algorithm any given top level decision vector is only evaluated once. All the generated sequences are stored for future comparison.

At the top level, the user must specify the planet to depart from $P_{dep}$, the final destination $P_{dest}$, the maximum number of phases allowed $n$, and the maximum number of deep space manoeuvres allowed per phase $DSM_{max}$. The top level decision vector is:

$$\mathbf{X} = [P_1, P_2, \ldots, P_{n-1}, D_1, D_2, \ldots, D_n] \tag{3.39}$$

To better illustrate this, consider following example. We want to find the optimal planetary sequence and combination of deep space manoeuvres for an Earth to Jupiter mission. A maximum of 5 phases are allowed, with no more that 2 deep space manoeuvres per phase. Set $P_{dep} = 3$, $P_{dest} = 5$, $n = 5$ and $DSM_{max} = 2$. This

produces the following decision vector

$$\mathbf{X} = [P_1, P_2, P_3, P_4, D_1, D_2, D_3, D_4, D_5] \tag{3.40}$$

where $P_{1..4} \in [0, 9]$ and $D_{1..5} \in [0, D_{max}]$. Intuitively it is possible to restrict the mission to the inner planets only by reducing the upper bound on $P$ to 5. Similarly one can implement a mission which contains no deep space manoeuvres at all by setting the upper bound on $D$ to 0.

By allowing the lower bound on the intermediate planet selection to be set to 0 we are able to consider missions with a varying number of phases. This is important because in general the mission analyst does not know in advance the optimal number of swingbys needed for a specific mission, and may otherwise have to perform multiple optimisations. In order to implement this some pre-processing is required. Before a sequence can be passed to the pruning algorithm the sub-vector $P$ must be examined using the following simple algorithm.

```
for i = 1:length(P)
   if P(i) == 0
      for j = i+1 : length(P)
         P(j) = 0
      end
   end
end
```

All the zeros are then stripped from $P$ as well as the corresponding number of elements from the sub-vector $D$ used to define the maximum number of deep space manoeuvres in a phase.

### Algorithm

The complete two level *hybrid* algorithm is described in this section. A mission analyst must characterize the mission by pre-selecting a few parameters at the top level:

- Launch window in MJD2000
- Departure planet, $P_{dep}$
- Destination body, $P_{dest}$
- Maximum number of phases, $n$
- Maximum number of DSM's per phase, $D_{max}$
- PSO swarm size, $S$
- Generations to run for, $N$

Once the top level settings have been chosen a few lower level settings need to be made at the objective function (previously described pruning algorithm) level. These parameters include

- Launch, DSM, swingby, insertion constraints
- DSM parameter bounds
- Insertion parameters
- Number of starts of the local optimiser in the first phase
- Number of starts of the local optimiser per window in subsequent phases

The algorithm then executes as described below. When line 12 is reached the pruning algorithm is called. It is here that the bulk of the processing is carried out. The notation used is as follows, $\mathbf{X}$ represents the vector of particles in the system, $\mathbf{V}$ represents the vector of associated velocities. $\mathbf{X}_g^*$ corresponds to the best globally known particle and $(X)_{pi}^*$ is the personal best solution of particle $i$.

*Hybrid Sequence optimisation Algorithm*

1. Initialise $\mathbf{X}$ uniformly randomly over the search space
2. For each particle vector $\mathbf{X}_i$
3. Round off each element of the particle to nearest whole number
4. end
5. Initialise $\mathbf{V}$ uniformly randomly over the search space
6. For each velocity vector $\mathbf{V}_i$
7. Round off each element of the vector to nearest whole number
8. end
9. Repeat
10. count = count + 1
11. For each population vector $\mathbf{X}_i$
12. $J_i = \text{prune}(\mathbf{X}_i)$
13. If $(J_i < J_{pi}^*)$
14. $J_{pi}^* = J_i$
15. $\mathbf{X}_{pi}^* = \mathbf{X}_i$
16. end
17. If $(J_i < J_g^*)$
18. $J_g^* = J_i$

19. $\mathbf{X}_g^* = \mathbf{X}_i$

20. end

21. $\mathbf{V}_i = round\{\omega \mathbf{V}_i + c_1 r_1(\mathbf{X}_p^* - \mathbf{X}_i) + c_1 r_2(\mathbf{X}_g^* - \mathbf{X}_i)\}$

22. $\mathbf{X}_i = round(\mathbf{X}_i + \mathbf{V}_i)$

23. Until count == $N$

**Complexity Analysis**

Consider equation (3.35), which gives an upper bound on the Lambert evaluations for the pruning stage followed by the global optimisation stage for a given sequence. Define

$$\beta = \gamma k_1 - k_2\Psi - 2k_3\Psi + (k_2 + k_3)\Psi n + (N_p N_i S + N_p N_g)(\bar{d} + 1)n \tag{3.41}$$

as such upper bound on the number of Lambert solver calls (both for pruning and global optimisation) for the most complex case considered by the upper level algorithm. Now suppose that the upper level algorithm has a a swarm of size $N_s$ and it is run for $N_u$ generations, then the total number of Lambert's solutions $L_s$ is upper bounded as follows:

$$L_s < N_s \beta N_u \tag{3.42}$$

Note that $\beta$ grows linearly with the number of phases $n$, so that the complexity of the sequence optimisation algorithm will also grow linearly with the number of phases.

### 3.3.4 Results

**Earth to Mercury**

A mission to send a space craft from Earth to Mercury is being considered. A maximum of 3 phases is allowed with no more than 1 DSM per phase. To aid the optimiser only planets between Mars and Mercury are valid options for this mission. A 2000 day launch window is permitted between 3000 and 5000 MJD2000 (19/3/08 - 9/9/13).

The problem is defined by the following parameters:

- Launch window: [3000 5000] MJD2000
- Departure planet $P_{dep} = 3$
- Destination $P_{dest} = 1$
- Maximum number of phases $n = 3$

- Maximum number of DSM's per phase $D_{max} = 1$
- Retrograde transfers: Not permitted
- Multiple revolutions: Not permitted
- PSO swarm size $S = 8$
- Generations to run for $N = 15$

As the swingby planets are not known, the time of flight bounds are implemented as look up tables. The time of flight bounds were mainly obtained by prior experience and trial and error. They were also setup to prevent multi-revolution orbits with only one deep space manoeuvre. The original GASP software was employed to make sure that suitable trajectories without deep space manoeuvres were accommodated for. Moreover, the time of flight intervals were kept small to aid the optimisation process. This mission only considers the planets Mars, Earth, Venus and Mercury, so the look up tables take the form of two $4 \times 4$ matrices shown below:

$$t_{of} \geq \begin{array}{c} \\ Me \\ E \\ V \\ M \end{array} \begin{array}{c} \begin{array}{cccc} Me & V & E & M \end{array} \\ \begin{pmatrix} 50 & 50 & 80 & 100 \\ 50 & 50 & 100 & 150 \\ 80 & 100 & 50 & 180 \\ 100 & 150 & 180 & 50 \end{pmatrix} \end{array} \tag{3.43}$$

$$t_{of} \leq \begin{array}{c} \\ Me \\ E \\ V \\ M \end{array} \begin{array}{c} \begin{array}{cccc} Me & V & E & M \end{array} \\ \begin{pmatrix} 300 & 300 & 300 & 350 \\ 300 & 350 & 400 & 300 \\ 300 & 400 & 400 & 450 \\ 300 & 300 & 450 & 400 \end{pmatrix} \end{array} \tag{3.44}$$

The parameters that characterise the deep space manoeuvre are bounded as usual so that $\theta \in [0, 2\pi]$, $\phi \in [-\pi/6, \pi/6]$ and $\alpha \in [0.1, 0.9]$. The parameter $r$ which defines the distance from the sun that the DSM takes place (in km) is bounded between the two semi-major axis' of the planets involved.

The best sequences found are shown in table 3.7. The global optimiser used at the end of the pruning stage was PSO with a swarm size of 35. Each solution family found was primarily optimised over 300 generations, the best of which was then optimised over 1000 generations. The notation used to describe DSM's in a phase is given by a string of values between 0 and $D_{max}$. For example a 3 phase mission with a DSM in the first and third phase only is represented by the string 101.

Looking at table 3.7 the best sequence appears to be E-E-V-Me, this sequence is present in the top 2 solutions. The difference being the presence of deep space manoeuvres. Carrying out a more thorough optimisation on the highest ranked solution involves allowing the optimiser to run for a larger number of generations. Of the four solution families found by the pruning algorithm for this particular mission, the low level decision variables for the optimal trajectory found are shown in table 3.8. The resulting deep space manoeuvres are characterised in table 3.9.

| Rank | Sequence | DSM | $\Delta V(km/s)$ |
|---|---|---|---|
| 1 | E-E-V-Me | 110 | 1.175 |
| 2 | E-E-V-Me | 111 | 1.746 |
| 3 | E-E-M-Me | 110 | 2.490 |
| 4 | E-V-Me | 01 | 2.838 |

Table 3.7: Highest ranked solutions for the Earth to Mercury sequence optimisation problem

| index | $t$ (MJD2000) | $T_{\text{of}}$ (days) | $r$ (km) | $\theta$ rad | $\phi$ rad | $\alpha$ |
|---|---|---|---|---|---|---|
| 0 | 4.9977e+003 | | | | | |
| 1 | | 498.4571 | 1.5150e+8 | -1.7949 | -1.3601e-4 | 0.5172 |
| 2 | | 460.2628 | 1.4010e+8 | -0.5011 | -0.0294 | 0.5384 |
| 3 | | 106.0772 | | | | |

Table 3.8: Low level decision vector values found after optimizing the pruned E-E-V-Me search space.

The trajectory described by the solution vector from table 3.8 is shown graphically in figure 3.11.

### Earth to Asteroid TW229

Asteroid TW229 was the subject of the first Global Trajectory optimisation Competition (GTOC) run by the ACT. The competition was designed to test global optimisation techniques on a technically demanding problem. For this reason we heave used Asteroid TW229 as the subject of this case. It is aimed to find the optimal sequence to reach the asteroid which minimises the total $\Delta V$. Note that the objective function used in the GTOC competition was different. The objective function used in the competition involved maximising the product of the final mass of the spacecraft and the absolute value of the scalar product of the relative velocity of the spacecraft with respect to the asteroid, and the heliocentric velocity of the asteroid.

The following parameters were used to optimise the sequence along with the standard DSM parameters decribed in the previous section:

- Launch window: [3000 5000] MJD2000
- Departure planet $P_{dep} = 3$
- Destination $P_{dest} = 53$
- Maximum number of phases $n = 4$
- Maximum number of DSM's per phase $D_{max} = 1$
- Retrograde transfers: Not permitted
- Multiple revolutions: Not permitted

| Manoeuvre | Type | Bodies Involved | $\Delta V(km/s)$ | Constraint Violation |
|---|---|---|---|---|
| 1 | Launch | E | 0.009471 | No |
| 2 | DSM | E-E | 0.005112 | No |
| 3 | DSM | E-V | 0.006725 | No |
| 4 | Swingby | E | 0.615045 | No |
| 5 | Swingby | V | 0.001380 | No |
| | | Total $\Delta V$ | 0.637734 | |

Table 3.9: E-E-V-Me mission's deep space manoeuvres resulting from the best low level solution vector

- PSO swarm size $S = 8$
- Generations to run for $N = 20$

| Rank | Sequence | DSM | $\Delta V(km/s)$ |
|---|---|---|---|
| 1 | E-E-Ast | 10 | 2.1998 |
| 2 | E-E-Ast | 01 | 2.7532 |
| 3 | E-V-Ast | 10 | 4.9765 |
| 4 | E-E-Ast | 11 | 5.0281 |
| 5 | E-A | 1 | 8.5676 |
| 6 | E-V-E-Ast | 110 | 14.4558 |
| 7 | E-V-J-Ast | 110 | 15.4633 |

Table 3.10: Highest ranked solutions for the Earth to Asteroid sequence optimisation problem

The solution vector for the highest ranked solution gives the following values $t_0 = 4723.9$, $T_{\text{of}}^{(1)} = 217.7497$, $r^{(1)} = 1.5158e+8$, $\theta^{(1)} = 2.9717$, $\phi^{(1)} = -1.0870e-4$, $\alpha(1) = 0.7837$ and $T_{\text{of}}^{(2)} = 676.2973$. This produces a launch velocity of 0.020716 km/s, the DSM in the resonant phase has a $\Delta V = 0.034745$ km/s and swingby $\Delta V = 2.144334$. The trajectory is show in figure 3.12.

Another interesting trajectory is the solution ranked third. This trajectory is plotted in figure 3.13 and described by the decision vector $t_0 = 4266.6$, $T_{\text{of}}^{(1)} = 413.2210$, $r^{(1)} = 1.1093e+8$, $\theta^{(1)} = -1.7980$, $\phi^{(1)} = -0.0536$, $\alpha(1) = 0.4398$ and $T_{\text{of}}^{(2)} = 403.4336$. The associated $\Delta V$'s are as follows; Launch = 3.762199 km/s, $\Delta V_{DSM}^{(1)} = 0.684771$ and $\Delta V_{ga}^{(1)} = 0.750823$.

## 3.4 Summary

In this part of the report a new method is presented for pruning the search space for the optimisation of multiple gravity assist trajectories with deep space manoeuvres and powered swingbys. The proposed pruning method extends the original GASP algorithm and samples each phase of the mission using constrained local optimisation, while a clustering technique is used to locate bounding boxes for the feasible

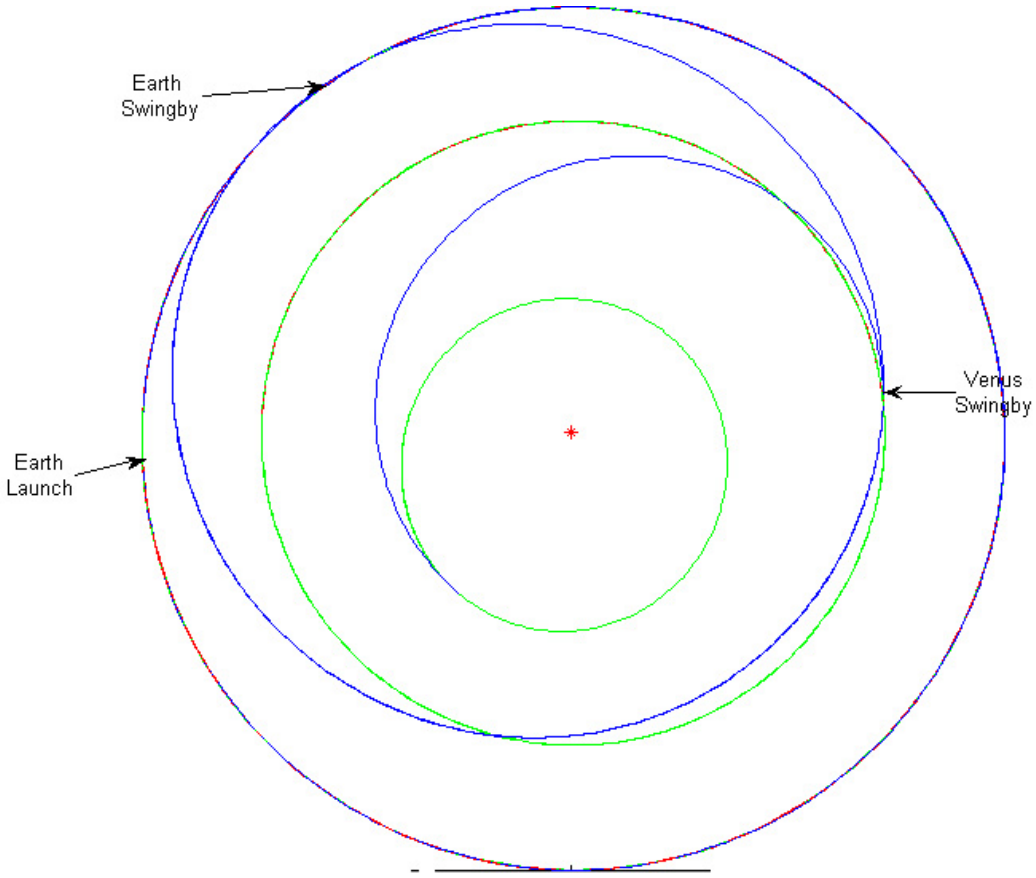



Figure 3.11: Earth-Earth-Venus-Mercury trajectory

regions of the search space. Furthermore, it has been shown that the complexity of the pruning algorithm followed by a global optimisation stage grows linearly with the number of phases present in the mission.

A two level algorithm for selecting discrete elements of a mission, such as the intermediate planetary sequence (given departure and arrival bodies), the number of deep space manoeuvres in each phase, and the presence of retrograde or multi-revolution phases. The upper level algorithm optimises the integer variables, and the use of a special version of Particle Swarm Optimisation is proposed for this purpose. At the lower level, each mission specified by the upper level is evaluated by a pruning stage followed by a global optimisation stage.

It has been shown that the complexity of the two level algorithm grows linearly with the maximum number of phases considered. However, given the CPU intensive nature of the lower level algorithm, the amount of computations for the sequence optimisation in realistic missions can be substantial. Depending on the mission and the parameters chosen the computations of the two level algorithm may take from several hours to a few days to complete.

Both the pruning method and the sequence optimisation technique have been tested with relevant case studies.

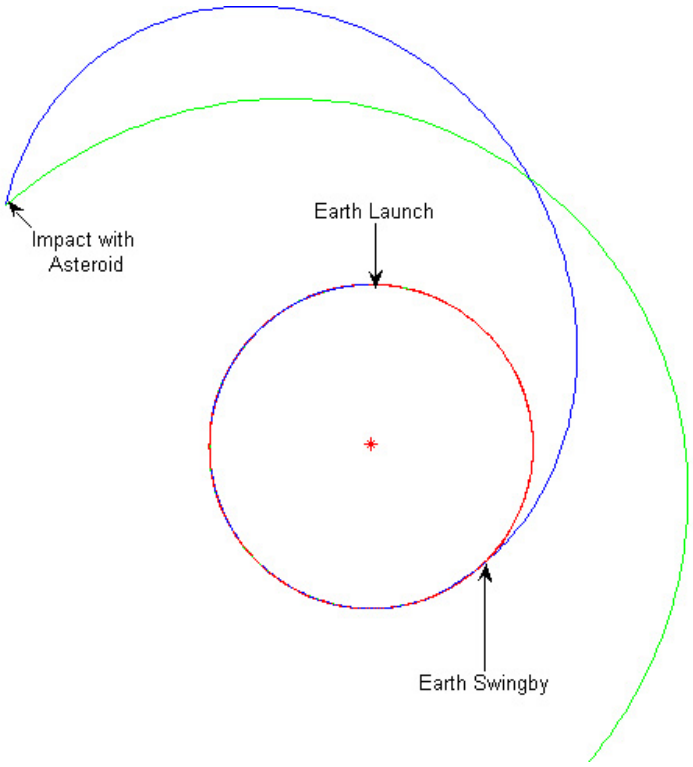



Figure 3.12: Earth-Earth-Asteroid trajectory

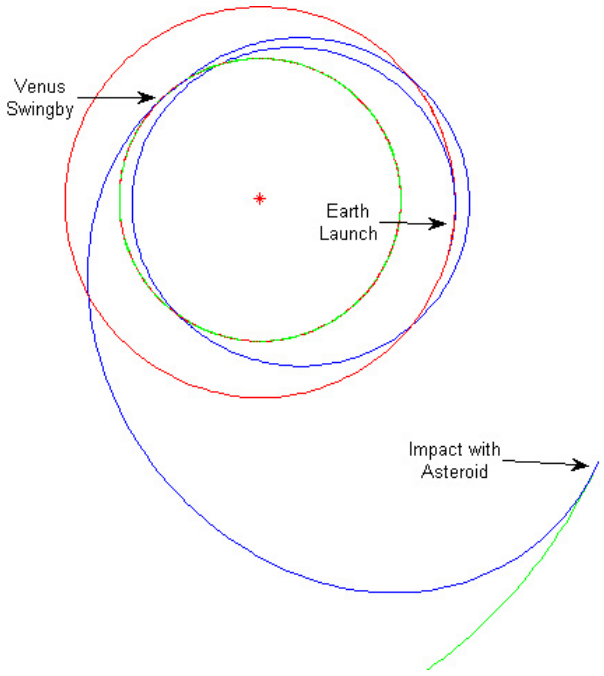



Figure 3.13: Earth-Venus-Asteroid trajectory

## 3.5 References


[1] D.R. Myatt, V.M. Becerra, S.J. Nasuto, and J.M. Bishop. Advanced global optimisation for mission analysis and design. Final report. ariadna id: 03/4101. contract number: 18138/04/nl/mv, European Space Agency, Noordwijk, The Netherlands, 2004.

[2] R. Storn and K. Price. Differential Evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimisation*, 11:341–359, 1997.

[3] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, 1995.

[4] D. Izzo, V. M. Becerra, D. R. Myatt, S. J. Nasuto, and J. M. Bishop. Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories. *Journal of Global Optimisation*, 38:283–296, 2007.

[5] V.M. Becerra, S.J. Nasuto, J. Anderson, M. Ceriotti, and C. Bombardelli. Search space pruning and global optimization of multiple gravity assist trajectories with deep space manoeuvres. In *IEEE Congress on Evolutionary Computation*, pages 957–964, Singapore, 2007.

[6] D. Comaniciu and P. Meer. Mean shfit: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[7] B. Finkston. Matlab implementation of the Mean Shift Clustering algorithm. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=10161&objectTy
2006.

[8] A.H. Aguirre, E.V. Diharce, and C.C. Coello. Constraint handling techniques for a non-parametric real-valued estimation distribution algorithm. In *IEEE Congress on Evolutionary Computation*, pages 654–661, Singapore, 2007.

[9] E.C. Laskari, K.E. Parsopoulos, and M.N. Vrahatis. Particle swarm optimization for integer programming. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1582–1587, 2002.