AUTOMATED TRAJECTORY PLANNING FOR MULTIPLE-FLYBY INTERPLANETARY MISSIONS

BY

JACOB ENGLANDER

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Professor Bruce A. Conway
Professor John E. Prussing
Professor Victoria L. Coverstone
Professor Anil N. Hirani

Abstract

Many space mission planning problems may be formulated as hybrid optimal control problems (HOCP), i.e. problems that include both real-valued variables and categorical variables. In interplanetary trajectory design problems the categorical variables will typically specify the sequence of planets at which to perform flybys, and the real-valued variables will represent the launch date, flight times between planets, magnitudes and directions of thrust, flyby altitudes, etc.

The contribution of this work is a framework for the autonomous optimization of multiple-flyby interplanetary trajectories. The trajectory design problem is converted into a HOCP with two nested loops: an "outer-loop" that finds the sequence of flybys and an "inner-loop" that optimizes the trajectory for each candidate flyby sequence. The problem of choosing a sequence of flybys is posed as an integer programming problem and solved using a genetic algorithm (GA). This is an especially difficult problem to solve because GAs normally operate on a fixed-length set of decision variables. Since in interplanetary trajectory design the number of flyby maneuvers is not known *a priori*, it was necessary to devise a method of parameterizing the problem such that the GA can evolve a variable-length sequence of flybys. A novel "null gene" transcription was developed to meet this need.

Then, for each candidate sequence of flybys, a trajectory must be found that visits each of the flyby targets and arrives at the final destination while optimizing some cost metric, such as minimizing Δv or maximizing the final mass of the spacecraft. Three different classes of trajectory are described in this work, each of which required

a different physical model and optimization method. The choice of a trajectory model and optimization method is especially challenging because of the nature of the hybrid optimal control problem. Because the trajectory optimization problem is generated in real time by the outer-loop, the inner-loop optimization algorithm cannot require any *a priori* information and must always return a solution. In addition, the upper and lower bounds on each decision variable cannot be chosen *a priori* by the user because the user has no way to know what problem will be solved. Instead a method of choosing upper and lower bounds via a set of simple rules was developed and used for all three types of trajectory optimization problem. Many optimization algorithms were tested and discarded until suitable algorithms were found for each type of trajectory.

The first class of trajectories use chemical propulsion and may only apply a Δv at the periapse of each flyby. These Multiple Gravity Assist (MGA) trajectories are optimized using a cooperative algorithm of Differential Evolution (DE) and Particle Swarm Optimization (PSO). The second class of trajectories, known as Multiple Gravity Assist with one Deep Space Maneuver (MGA-DSM), also use chemical propulsion but instead of maneuvering at the periapse of each flyby as in the MGA case a maneuver is applied at a free point along each planet-to-planet arc, i.e. there is one maneuver for each pair of flybys. MGA-DSM trajectories are parameterized by more variables than MGA trajectories, and so the cooperative algorithm of DE and PSO that was used to optimize MGA trajectories was found to be less effective when applied to MGA-DSM. Instead, either PSO or DE alone were found to be more effective.

The third class of trajectories addressed in this work are those using continuous-thrust propulsion. Continuous-thrust trajectory optimization problems are more challenging than impulsive-thrust problems because the control variables are a continuous time series rather than a small set of parameters and because the spacecraft

does not follow a conic section trajectory, leading to a large number of nonlinear constraints that must be satisfied to ensure that the spacecraft obeys the equations of motion. Many models and optimization algorithms were applied including direct transcription with nonlinear programming (DTNLP), the inverse-polynomial shape-based method, and feasible region analysis. However the only physical model and optimization method that proved reliable enough were the Sims-Flanagan transcription coupled with a nonlinear programming solver and the monotonic basin hopping (MBH) global search heuristic.

The methods developed here are demonstrated to optimize a set of example trajectories, including a recreation of the Cassini mission, a Galileo-like mission, and conceptual continuous-thrust missions to Jupiter, Mercury, and Uranus.

To my family past, present, and future.

Acknowledgments

This work was made possible by the support of several individuals and groups. I thank first my advisor, Bruce Conway, who guided me through the research and writing process these past seven years. Next, my committee members, John Prussing, Victoria Coverstone, and Anil Hirani who motivated and supported me with their interest in my work. Financial support for this work was provided by the Navigation and Mission Design Branch at NASA Goddard Space Flight Center via a Graduate Student Researchers Program (GSRP) Fellowship. In addition, several colleagues at Goddard, notably Steve Cooley, John Downing, Brent Barbee, Dave Folta, Trevor Williams, and Steve Hughes showed special interest in this work and helped me through many challenges in both optimization and modeling. Thank you to my fellow students at Illinois, notably Alex Ghosh, Chris Martin, Bradley Wall, Christian Chilan, Pradipto Ghosh, Brianna Aubin, Donald Ellison, and Joan Stupik, for sharing their ideas, catching my mistakes, and making me think critically about everything that I did. Thank you to my father, Arnold Englander, whose interest in stochastic optimization motivated the creation of Cauchy Monotonic Basin Hopping, and whose graduate studies in genetic algorithms inspired me as an infant. Thank you to my mother, Karen Englander, who always assumed that there was nothing that I could not do. Thank you to my stepmother, Joyce Cotter, for reminding me when I needed to hear it that graduate school is a process that must eventually end with a product. Thank you to my grandfather, Jerome Fischer, who set me on the path to become an aerospace engineer when I was only five years old. Thank you espe-

cially to my fiancée, Tiffany Moldenhauer, for her unwavering emotional support and the perspective of a veterinarian and biochemist, without which development of the null-gene transcription would have been much more difficult. Thank you to my best friend Michael Brothers for his support and chemical biologist's perspective. Thank you to Clementine, Curtis, and Cinnamon for their enthusiasm for this project and writing process, especially the parts that involved almonds, dandelions, strawberries, and tummy rubs. And finally, a hearty thank you to the men and women of the Illinois Cross Country and Track Clubs, who reminded me every day that nothing important can be done alone.

Table of Contents

Nomenclature

$\mathbf{q}$	Ordered vector of events in an HOCP
$\mathbf{s_{mb}}$	spacecraft state at the match point propagating backward
$\mathbf{s_{mf}}$	spacecraft state at the match point propagating forward
$\mathbf{v_{\infty/out}}$	outgoing velocity of a spacecraft at a flyby, relative to the planet
$\mathbf{v_{\infty/in}}$	incoming velocity of a spacecraft at a flyby, relative to the planet
$\mathbf{v_{pl}}$	velocity of a planet relative to the sun
$\mathbf{v_{sc/in}}$	velocity of a spacecraft incoming to a flyby, relative to the sun
$\mathbf{v_{sc/out}}$	velocity of a spacecraft outgoing from a flyby, relative to the sun
ΔV	Change in velocity due to a maneuver or set of maneuvers
$\Delta v_{insertion}$	change in velocity necessary to insert into a planet-centered orbit
Δv_{LV}	change in velocity provided by the launch vehicle
δ	flyby turn angle
η	burn index for MGA-DSM missions
γ	b-plane insertion angle
$\hat{i}$	unit vector

$\hat{j}$	unit vector
$\hat{k}$	unit vector
ν	for PSO, velocity of a particle in the decision space
$\mathbf{d}$	difference vector for differential evolution
$\mathbf{P}$	for DE or PSO, a population of decision vectors
$\mathbf{P}_\nu$	for PSO, a population of velocity vectors in the decision space
$\mathbf{r}_{pl}$	position vector of a planet
$\mathbf{r}_{s/c}$	position vector of a planet
$\mathbf{u}$	a decision vector
$\mathbf{u}_{best}$	best known decision vector
$\mathbf{u}_{global-best}$	for PSO, the best point found by the swarm
$\mathbf{u}_{i-local-best}$	for PSO, the best point found by the ith particle
$\mathbf{u}_{trial}$	trial decision vector
$\mathbf{x}$	decision vector
$\mathbf{x}_{lb}$	lower bounds on the decision vector
$\mathbf{x}_{ub}$	upper bounds on the decision vector
μ_{pl}	gravitational parameter of a planet
μ_{GA}	GA mutation probability
ν	interior point control variable used in DTNLP
ν_{max}	for PSO, the maximum allowed velocity in the decision space

ϕ	angle defining motion out of the ecliptic plane
ρ	a random number
σ	maximum step size for standard Monotonic Basin Hopping
τ	orbital period
θ	angle defining motion in the ecliptic plane
$\widetilde{\mathbf{a}}$	derivative of the velocity vector in the direction of increasing θ
$\widetilde{\mathbf{v}}$	vector derivative of the position vector in the direction of increasing θ
A	matrix describing the linear constraints in an NLP problem
$a_{desired}$	semi-major axis of the desired orbit
$c(\mathbf{x})$	nonlinear constraint function
CR	for GA or DE, crossover ratio
CR_{GA}	GA crossover ratio
D	thruster duty cycle
DLA	declination of launch asymptote
e_{in}	eccentricity on the incoming leg of a flyby
e_{out}	eccentricity on the outgoing leg of a flyby
F	for DE, scaling factor
$f(\mathbf{x})$	objective function for optimization
$g(\mathbf{x})$	penalty function for enforcing constraints in evolutionary optimization
g_0	Earth gravity at sea level

h	length of one time step
I_{sp}	Specific impulse
k_time	flight time penalty function scaling constant
k_{flyby}	flyby orbit energy penalty function scaling constant
$m_\odot$	mass of the sun
m_{final}	final mass of the spacecraft
m_{max}	maximum allowed mass of the spacecraft
m_{pl}	mass of a planet
N	number of time steps in a phase
n	dimensionality of the decision space
n_c	number of cities in the MTSP
N_p	GA population size
N_q	For HOCP, number of discrete events in Q
n_T	number of thrusters
$N_{\text{not improve}}$	for MBH, number of failed attempts until the algorithm resets
N_{elite}	GA elite count
N_{gen}	Maximum number of GA generations
$N_{parents}$	Size of GA parent pool
$N_{stall-gen}$	Maximum number of GA stall generations
$P(r)$	available electric power as a function of distance from the sun

P_{1AU}	available power at a distance of 1 AU from the sun
Q	Set of discrete events that can take place in an HOCP
q	Discrete event chosen in an HOCP
$r_{\odot}$	radius of the sun
R_p	flyby periapse ratio
r_p	flyby periapse distance
r_{pl}	radius of a planet
r_{SOI}	radius of a planet's gravitational sphere of influence
RLA	right ascension of launch asymptote
T_i	flight time for the ith phase
$T_{flight-max}$	maximum time of flight
T_{launch}	launch epoch
u	generalized control variable used in DTNLP
$v_{p/desired}$	velocity of the spacecraft at periapse of the desired orbit
$v_{p/in}$	velocity of the spacecraft at periapse of an incoming hyperbola
w_1	for PSO, the weighting factor on the intertial term
w_2	for PSO, the weighting factor on the cognitive term
w_3	for PSO, the weighting factor on the social term
B&B	Branch and Bound
COOP	Cooperative algorithm of DE and PSO

DCNLP	Direct Collocation with Nonlinear Programming
DE	Differential Evolution
DTNLP	Direct Transcription with Nonlinear Programming
EMTG	Evolutionary Mission Trajectory Generator
EMTG-LT	Evolutionary Mission Trajectory Generator - Low-Thrust
G	universal gravitation constant
HOCP	Hybrid Optimal Control Problem
JIMO	Jupiter Icy Moons Orbiter
MBH	Monotonic Basin Hopping
MBH-C	Monotonic Basin Hopping with Cauchy hops
MGA	Multiple Gravity Assist
MGA-DSM	Multiple Gravity Assist with one Deep Space Maneuver
MGA-LT	Multiple Gravity Assist with Low-Thrust
MTSP	Motorized Traveling Salesman Problem
MTSP	Motorized Traveling Salesman Problem
NLP	Nonlinear Programming
PSO	Particle Swarm Optimization
TSP	Traveling Salesman Problem

Chapter 1

Introduction

1.1 An Introduction to Interplanetary Trajectories

The age of interplanetary exploration began in 1962 with the success of Mariner 2 to Venus [1]. While early study of other planets was confined to observation by telescope, the advent of spaceflight allowed scientists to send instruments directly to the planets and observe them *in situ*. The early interplanetary missions were flown to Earth's immediate neighbors: first Venus, then Mars. The total change in velocity, or ΔV, necessary to travel to Venus or Mars is low enough that direct flights were possible using the engine technology of the 1960s. Unfortunately travel to more distant destinations such as Mercury, Jupiter, Saturn, or most comets and asteroids requires a larger ΔV and therefore more propellant. However it is not always feasible to provide this propellant, and so many destinations were not reachable by direct trajectories.

The most obvious solution to the problem of having insufficient ΔV capability to reach more distant destinations is to equip the spacecraft with a more effective engine. A higher specific impulse, or I_{sp}, means that a spacecraft will use less propellant to achieve the same ΔV or the same amount of propellant to achieve a much higher ΔV. Unfortunately chemical thrusters, whose I_{sp}'s range from 200 to 470 seconds depending on the type of propellants being reacted and the configuration of the thruster, are not sufficient to enable direct flights to distant destinations.

Electric thrusters have much greater efficiencies, with I_{sp} of 2000 seconds or greater. There are many types of electric thruster, but all rely on the ejection of a stream of charged particles out the back of the spacecraft. Since the I_{sp} of a thruster is directly correlated to the exhaust velocity, electric engines can be arbitrarily efficient. One must only increase the speed of the charged particles. Unfortunately in order to do so, one must either use very light-weight particles and therefore reduce the thrust of the engine, or one must supply a great deal of electrical power. In practice, electric thrusters tend to have high I_{sp} but low thrust, and require a great deal of electrical power. There are two general classes of electric propulsion system. The first are fixed-power systems; systems that supply a fixed I_{sp} and thrust throughout the mission. Two types of fixed-power systems are nuclear-electric propulsion (NEP), which uses a nuclear reactor to power the electric thruster and radioisotope-electric propulsion (REP), which uses a radioisotope thermal generator (RTG) to power the thruster by radioactive decay. To date, no NEP or REP spacecraft have been flown but both have been considered for missions in the past [2, 3].

The second type of electric propulsion system is the solar electric propulsion system (SEP). SEP spacecraft use a set of solar arrays to power their thrusters. Because available solar power diminishes as the spacecraft moves farther from the sun, SEP spacecraft become less capable of maneuvering as they move away from the sun and eventually the SEP system becomes useless. However SEP is very effective in the inner solar system. Many SEP missions have been flown, including Deep Space 1 [4], Hayabusa [5], DAWN [6], and many Earth-orbiting satellites. Unfortunately SEP systems tend to have very low thrust, and so SEP missions tend to take more time to get to their destination than a comparable chemical mission. REP missions would suffer from the same problem due to the very small amount of electrical power available from a radioisotope thermal generator. Electric propulsion systems have been demonstrated with I_{sp} as high as 4100 seconds. Because they operate for long periods

of time, electric propulsion systems, as well as any other system that operates for a long period, are often called "continuous thrust" propulsion.

Interplanetary SEP systems were not available until the 1990s [4]. Another solution to the problem of reaching distant destinations was necessary. The solution was the planetary flyby, or gravity assist maneuver, discovered in 1961 by Michael Minovitch, a researcher at the Jet Propulsion Laboratory [7]. Minovitch observed that a close encounter between a spacecraft and a planet results in a momentum exchange. Since the planet is large and the spacecraft is relatively tiny, there is no noticeable effect on the planet. The flyby changes the direction, although not the magnitude, of the spacecraft's velocity relative to the planet. This in turn changes the velocity of the spacecraft relative to the sun. The change in direction can be dramatic. Mission designers can achieve a wide range of course changes by varying the velocity at which the spacecraft approaches the planet as well as the point of closest approach. A sample planetary flyby maneuver is shown in Figure 1.1.

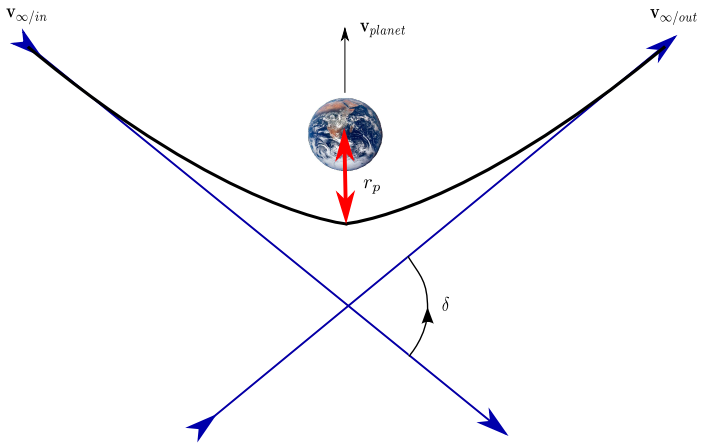


Figure 1.1: Example of a Planetary Flyby Maneuver

Flyby maneuvers have been used in interplanetary missions since Mariner 10, which launched in 1964 and visited Mercury via multiple flybys of Venus. Other notable missions employing multiple flybys include Galileo, which traveled to Jupiter in 1989 following flybys at Venus and Earth [8, 9], Cassini, which traveled to Sat-

urn in 1997 via flybys of Venus, Earth, and Jupiter [10], and MESSENGER, which traveled to Mercury in 2004 via flybys of Earth and Venus [11]. A diagram of the Galileo interplanetary trajectory is shown in Figure 1.2. Two missions are currently *en route* to their destinations using planetary flybys: Rosetta, which launched to 67P/Churyumov-Gerasimenko in 2004 via flybys of Earth and Mars [12], and New Horizons, which launched to Pluto and the Kuiper Belt in 2006 via flybys of Mars and Jupiter [13].

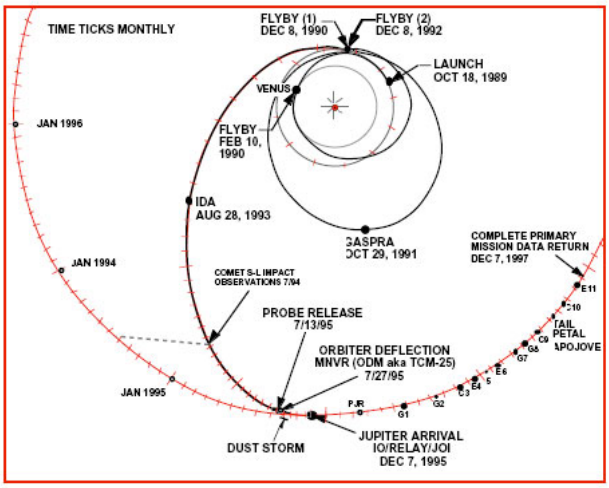


Figure 1.2: Galileo Interplanetary Trajectory [9]

1.2 Designing Interplanetary Trajectories

The problem of designing multiple-flyby interplanetary trajectories is very challenging. The mission designer must choose the sequence of planets to be visited for flybys, the dates of those encounters, and the spacecraft trajectory that reaches the targets on the chosen dates and optimizes some cost function, typically either maximizing mass delivered to the final destination or minimizing the total ΔV for the mission. To create such a trajectory, the designer must create a sequence of coast arcs and propulsive maneuvers that may be impulsive, i.e. nearly instantaneous changes in

velocity for missions employing chemical propulsion, or long continuous thrust arcs for missions employing electric propulsion. This process is complex and many feasible solutions may exist for each problem. Because of the complexity of multiple flyby interplanetary trajectory problems, it is impractical to design such missions "by hand." Many researchers have therefore sought to automate part or all of the mission design process.

1.2.1 Impulsive-Thrust Interplanetary Trajectory Design

Much work has been done on the optimization of trajectories using impulsive thrust and a specified flyby sequence, that is, a sequence chosen by the designer *a priori*. In this case, the problem of interest is to determine the optimal encounter dates, flyby close approach points, and the dates, magnitudes, and directions of any deep-space maneuvers. Several researchers have developed methods to solve this class of problem. The least complex type of multiple flyby, impulsive-thrust problem is the multiple gravity-assist, or MGA, trajectory. In an MGA trajectory, the spacecraft follows a ballistic path from planet to planet and impulsive maneuvers are allowed only at the periapse, or closest point to the planet, of each flyby. This greatly reduces the number of variables in the problem. The MGA model was first described by Vasile and De Pascale in 2006 [14], but the term MGA was not coined until 2008 by Vinkó and Izzo [15]. However, the constraint to perform impulsive maneuvers only at flyby periapse is not realistic; the optimal location for a maneuver is often not the periapse of a flyby and no such maneuver has ever been performed in a real mission.

Later, the multiple gravity-assist with one deep-space maneuver, or MGA-DSM, model was developed by Vinkó and Izzo [15]. In MGA-DSM, the spacecraft may perform one deep-space maneuver, or DSM, at any point along each planet-to-planet arc. No burns at flyby periapse are permitted. Several global optimization schemes, including genetic algorithms (GA), differential evolution (DE), particle swarm opti-

mization (PSO), monotonic basin hopping (MBH), inflationary differential evolution (IDEA), ant colony optimization (ACO), and cooperative algorithms composed of several of the preceding methods, have been used to solve MGA and MGA-DSM problems [14–17].

In addition, Vasile *et al.* developed a method called TACO that uses ant colony optimization to build trajectories with no deep-space maneuvers [18]. Finally, Olds, Kluever, and Cupples developed a tool called MDTOP that uses differential evolution along with a trajectory model that is similar to but more detailed than MGA-DSM to design trajectories with deep-space maneuvers and realistic launch and arrival models [19].

1.2.2 Continuous-Thrust Interplanetary Trajectory Design

Additionally, much work has been devoted to the problem of finding optimal continuous-thrust trajectories for problems with no flybys or when the flyby sequence is specified. The continuous-thrust problem is qualitatively different from the impulsive problem. The impulsive problem is a parameter optimization problem where the optimizer must choose a small number of discrete parameters such as launch date, flight time, flyby altitude, etc. The continuous-thrust problem contains both discrete parameters and also a time history of control variables, typically thrust direction and magnitude history, which the optimizer must find.

There are two families of approaches to this problem: direct and indirect. In indirect trajectory optimization, the trajectory design is formulated as an optimal control problem and analytic necessary and sufficient conditions are derived in terms of the system states and Lagrange multipliers, which represent the sensitivity of the solution to the constraints. The problem is then solved as a multi-point boundary value problem [20]. Some indirect optimization problems are small enough, in terms of the number of variables and constraints, that they may be solved analytically.

However spacecraft trajectory optimization problems usually have a large number of variables and constraints and therefore must be solved numerically. A nonlinear programming (NLP) problem solver may be used to find the optimal value of the Lagrange multipliers. However, NLP problem solvers require a good initial guess and cannot converge to a solution unless the initial guess is adequate. Small variations in the initial guess of the Lagrange multipliers may represent a significantly different trajectory, and so indirect optimizers are very sensitive to the choice of initial guess. Choosing a good guess can be very non-intuitive.

In direct optimization, the problem is parameterized by physically meaningful decision variables such as flight times, thrust magnitudes and control angles, flyby altitudes, etc. rather than by Lagrange multipliers. Many different transcription methods are used with direct solvers, including direct collocation with a polynomial approximation to the control parameters [21], direct transcription [22], perturbative methods [23], and approximation of the continuous trajectory as a series of small coast arcs punctuated by small impulses [24]. Like indirect methods, direct methods require a NLP problem solver and a good initial guess of the decision variables.

1.2.3 Initial Guess Generation for the Continuous-Thrust Interplanetary Trajectory Design Problem

Regardless of whether an indirect or direct optimization method is chosen, a good initial guess of the decision variables must be supplied. Sometimes this can be done with a very simple intuitive method. For example, Enright and Conway [21] solved the problem of sending a continuous-thrust spacecraft to an escape trajectory with minimum thrusting time using an initial guess where the spacecraft simply thrusts along its velocity vector. For more complex problems, such as when the spacecraft needs to match position and velocity (rendezvous) with a body or perform multiple flybys, a more sophisticated initial guess is required.

Initial guesses are often generated by solving an approximated version of the continuous-thrust trajectory optimization problem that sacrifices fidelity in exchange for reducing the number of variables and constraints in the problem so that it may be solved by intuition or by stochastic methods. For example, one can sometimes use an impulsive-thrust solution as an initial guess for the continuous-thrust optimization [25].

Another popular approach is to approximate the trajectory with a shape function that can be parameterized by a small number of variables, and then find the optimal values of those variables using a stochastic solver such as an evolutionary algorithm. One can then analytically derive what the time history of the control variables would be in order to achieve that trajectory, and then use the derived controls as the initial guess for the original trajectory optimization problem. For example, Petropoulos and Longuski developed a shape-based method where the trajectory is approximated as an exponential sinusoid curve [26]. Other shape-based methods were developed by Wall and Conway, who modeled the trajectory as an inverse polynomial [27], and Novak and Vasile, who developed a shape-based method in spherical coordinates [28]. All three methods are used with evolutionary algorithms to choose the optimal shape-based trajectory approximation. These shaped based methods can be used to quickly generate an initial guess for the trajectory optimization problem, but unfortunately all share the deficiency that no constraints can be imposed on the control variables. Often a trajectory is generated that appears to be optimal in terms of the supplied objective function but in fact requires an infeasible amount of thrust.

1.2.4 Autonomous Solutions to the Continuous-Thrust Trajectory Optimization Problem

Several researchers have explored methods of solving continuous-thrust trajectory optimization problems using a fully autonomous approach, i.e. one that produces its

own initial guess or avoids the use of an initial guess altogether. Chilan and Conway [29] developed two methods, Feasible Region Analysis (FRA) and Conditional Penalty (CP), both of which use a binary genetic algorithm to choose a sequence of thrust and coast arcs, and then a real-valued genetic algorithm along with a direct transcription method to find the optimal trajectory. Both of these methods are designed to always find a mathematically feasible solution. This is necessary because they are designed for use in a hybrid optimal control problem (HOCP) framework with two nested optimization loops. The outer-loop depends on the solution to the inner-loop problem, and therefore may only work if the inner-loop always returns a solution and never simply halts. Ghosh and Conway [30] used PSO to solve a continuous-thrust orbit transfer problem. However, their approach used a penalty function to enforce the nonlinear constraints. The penalty function contained scaling factors that had to be chosen by experiment, making their method unsuitable for use in an autonomous mission design algorithm.

Coverstone *et al.* used a GA to choose initial guesses for the low-thrust trajectory optimization package SEPTOP, which uses an indirect transcription with an NLP solver [31, 32]. Later, Vavrina and Howell [33] used a GA to choose initial values for their GALLOP program, which uses the Sims-Flanagan transcription method along with an NLP solver. Most recently, Yam, Di Lorenzo, and Izzo [34] used monotonic basin hopping (MBH) along with the Sims-Flanagan transcription to quickly and reliably solve continuous-thrust problems. All three of these methods have been demonstrated on problems involving multiple flybys as well as continuous-thrust propulsion, although all require a fixed flyby sequence.

1.2.5 Automated Choice of the Flyby Sequence

Several researchers have addressed the problem of finding the optimal sequence of planets for a multiple-flyby mission, although most previous works have focused on

missions that use only impulsive-thrust. Jet Propulsion Lab's S-TOUR software [35] solves for all possible sequences of flybys using an exhaustive grid search - an effective, albeit computationally expensive - method of finding the optimal sequence of flybys for a mission with impulsive-thrust. Ceriotti and Vasile [36] developed a method to find multiple-flyby trajectories with deep space maneuvers using ant colony optimization. Gad and Abdelklalik [37, 38] developed two methods that use a genetic algorithm (GA) to choose both the optimal flyby sequence and the optimal trajectory for that sequence.

Finally, other researchers have developed methods of autonomously designing low-thrust trajectories, including choosing the optimal sequence of bodies (planets, moons, asteroids, and comets) to visit and then finding the optimal trajectory. Braun and Alemany solved a multiple-asteroid rendezvous problem by using a binary genetic algorithm to choose a sequence of asteroids [39], along with Jet Propulsion Lab's MALTO trajectory optimizer that uses the Sims-Flanagan method to find the optimal trajectory. Additionally, Vasile and Campagnola designed a mission to Europa by approximating the low-thrust transfer between two planets as a Hohmann transfer, and then using a branch and bound method, as described in Section 3.2 to find feasible sequences of flybys [25]. A direct method based on the Finite Elements in Time approach [23] was then used with an NLP solver to find the optimal trajectory for each sequence.

However, to date no fully automated method has been demonstrated to optimize complex multiple flyby missions with impulsive propulsion, such as Cassini, with no *a priori* information about the sequence of planets. Similarly, no existing fully automated method can optimize multiple flyby missions with continuous-thrust propulsion. In this work, the multiple-flyby trajectory optimization problem is formulated as a Hybrid Optimal Control Problem (HOCP) with two nested solvers. The "outer-loop" solver chooses the optimal sequence of flybys using an integer genetic algorithm

(GA). For each candidate sequence of flybys, an "inner-loop" trajectory optimization problem must be solved to find the optimal launch epoch, encounter epochs, flyby altitudes, maneuver epochs, magnitudes, and directions, as well as several other parameters that define the trajectory. The resulting tool, referred to as the Evolutionary Mission Trajectory Generator (EMTG), is capable of autonomously designing multiple gravity-assist trajectories with both impulsive and continuous-thrust. The EMTG is demonstrated on several problems including a reproduction of the Cassini mission and proposed missions to Uranus and Mercury.

1.3 Thesis Outline

Chapter 2 provides an overview of the Hybrid Optimal Control (HOCP) process. A simple HOCP example, the motorized traveling salesman problem (MTSP) is described. The HOCP model used in this work is then explained.

Chapter 3 describes the outer-loop transcription method and solver that are used to find the optimal sequence of flybys. A novel transcription method using "null genes" is introduced, and the genetic algorithm (GA) used to solve the outer-loop problem is described.

Chapter 4 describes the inner-loop modeling and solution methods for impulsive-thrust probems. Two types of impulsive-thrust mission models are described. The multiple gravity-assist, or MGA, model allows impulsive rocket burns only at periapse of a flyby. The multiple gravity-assist with one deep space maneuver, or MGA-DSM, model extends MGA by allowing a single impulsive rocket burn at any point along a planet-to-planet phase. The modeling of additional constraints such as bounds on flight time and arrival date and the prevention of collisions with planets is discussed. Then, the solution methods used for impulsive-thrust problems are described. Finally, a set of laws are introduced that dictate the automated choosing of bounds on the

inner-loop decision parameters.

Chapter 5 presents examples of impulsive-thrust problems. Two missions are optimized using the MGA model: a Galileo-like mission to Jupiter and a Cassini-like mission to Saturn. One mission is optimized using the MGA-DSM model: a mission to Saturn. The optimal MGA-DSM mission to Saturn is almost identical to the actual Cassini mission.

Chapter 6 describes the methods used to model and solve continuous-thrust problems. Several models are presented: Direct Transcription with 4th-Order Runge Kutta Integration (DTRK), Feasible Region Analysis (FRA), the Inverse Polynomial Shape-Based Method (IPSBM), and the Sims-Flanagan transcription (SF). The Sims-Flanagan transcription is then incorporated into the Multiple Gravity Assist with Low-Thrust (MGA-LT) model. Next, optimization methods are discussed. Finally, just as for the impulsive methods described in Chapter 4, a set of laws are introduced that dictate the automated choosing of bounds on the inner-loop decision parameters.

Chapter 7 presents examples of continuous-thrust problems. First, a nuclear electric propulsion (NEP) mission to Jupiter, similar to the canceled Jupiter Icy Moons Orbiter (JIMO) is optimized. Next, a mission to Mercury, similar to the European Space Agency's upcoming BepiColombo is created. Finally, a hypothetical mission to Uranus using solar-electric propulsion (SEP) is optimized.

Finally, Chapter 8 presents a summary of all work accomplished and recommendations for future work.

Chapter 2

Hybrid Optimal Control

2.1 Introduction

The interplanetary mission planning problem is a member of a general class of problems that include both discrete and continuous variables, called "Hybrid Optimal Control Problems," or HOCPs. In a HOCP, the choice of discrete variables is used to define the problem parameterized by the continuous variables. A classic example is the Motorized Traveling Salesman Problem (MTSP) [29, 40, 41], in which a salesman must visit each of several cities. The salesman drives a car that can accelerate and turn. The objective is to choose both the order of cities to be visited (the discrete variables) and the time-history of steering and acceleration controls (the continuous variables) that guide the car to each destination in minimum time.

A very effective method of solving HOCPs is the method of nested optimization loops. An outer-loop solver is used to choose a set of discrete events, which are then used to construct a inner-loop problem defined by continuous variables. The advantage of this approach is that two separate solvers, each ideally suited to the problem type, may be used. The outer-loop is an integer programming problem best addressed with an integer-valued global optimization technique such as a Genetic Algorithm (GA) [29, 41], Branch and Bound (B&B) [29, 41, 42], or for small problems total enumeration [40]. The inner-loop is a real-valued optimization problem best addressed with evolutionary algorithms (EAs) such as a real-valued GA, Differential Evolution (DE), Particle Swarm Optimization(PSO), or a Nonlinear Programming

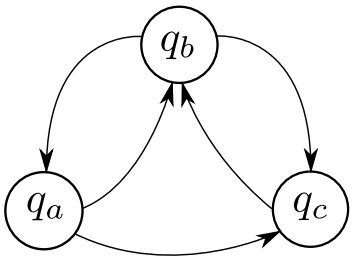


Figure 2.1: Example of a Digraph

(NLP) solver.

In this chapter, the HOCP framework introduced by Ross and D'Souza is described [42]. Next, the classic Motorized Traveling Salesman Problem (MTSP) is used as an illustrative example of how an HOCP is constructed. Finally, the specific HOCP structure used in this work is described.

2.2 Formal Definition of Hybrid Optimal Control

Ross and D'Souza [42] created a formal definition of the generalized HOCP. They define a set Q of discrete, or qualitative, states that the HOCP may occupy. For example, in the MTSP, Q is a set of cities that may be visited. The mission planner, in this case the outer-loop solver, must choose an ordered vector $\mathbf{q}$ of events $q \in Q$ that define the optimal sequence of events. The length of $\mathbf{q}$ may be either fixed or variable, although in the example presented by Ross and D'Souza the length of $\mathbf{q}$ if fixed. Each candidate $\mathbf{q}$ is used to define an inner-loop optimization problem that must then be solved in order to associate a cost function value with $\mathbf{q}$. In many HOCPs, certain events q_i can only follow certain other events. These relationships are best represented in a directed graph, or digraph, as shown in Figure 2.1. In a digraph, the nodes represent events and the directed edges show the allowed transitions between events.

When writing HOCP computer programs, it is convenient to write the digraph mathematically as a "switching set" $S(q_i, q_k)$ [42]. If $S(q_i, q_k) \neq \emptyset$, then the transition

between q_i and q_k is allowed. The possible switching sets may be encoded in an $N_q \times N_q$ switching matrix, where N_q is the number of states in Q, i.e.:

$$A_{ij} = \begin{cases} 1 & \text{if } S(q_i, q_k) \neq \emptyset \\ 0 & \text{if } S(q_i, q_k) = \emptyset \end{cases} \tag{2.1}$$

For example, in Figure 2.1, q_b and q_c are accessible from q_a, q_a and q_c are accessible from q_b, but only q_b is accessible from q_c. It is not permissible to remain in the same state for more than one time step. Therefore,

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

The task of the finite state outer-loop automaton is to search the space of sequences $\mathbf{q}$, which are composed of elements of Q, and obey the switching constraints defined in the switching matrix. Each candidate $\mathbf{q}$ has an associated value, found by optimizing a cost functional, $f(\mathbf{x_q}, \mathbf{q})$, where $\mathbf{x_q}$ is a real-valued vector defining the trajectory that travels through the sequence $\mathbf{q}$. Assuming that the inner-loop solver responsible for finding the optimal value of x is successful, then the outer-loop will be able to choose an optimal sequence $\mathbf{q}^*$. The choice of $\mathbf{q}^*$ is an integer programming problem, which may be solved by a variety of methods including Branch and Bound (B&B) [40, 43] and Genetic Algorithms (GA) [29, 41, 44].

2.3 Motorized Traveling Salesman Problem

A classic example of an HOCP is the Motorized Traveling Salesman Problem (MTSP). The MTSP, first defined by von Stryk [40], is a variant of the classic Traveling Salesman Problem (TSP) where a salesman must visit each of a set of n_c cities once in

minimum time. In the TSP, the travel time between each pair of cities is fixed. However in the MTSP, the cities are given coordinates on the XY plane and the salesman drives a car with velocity v and steering angle β. The control variables are the acceleration α and the steering rate γ. The differential equations defining the motion of the salesman's car are:

$$\begin{aligned} \dot{x} &= v cos\,(\beta) \\ \dot{y} &= v sin\,(\beta) \\ \dot{v} &= \alpha \\ \dot{\beta} &= \gamma \end{aligned} \tag{2.3}$$

with bounded controls $|\alpha| \leq 1$ and $|\gamma| \leq 1$.

von Stryk and Glocker [40] solved a version of the MTSP with $n_c = 3$. In this version of the problem, there are 6 possible sequences among the three cities. However, since the sequences may be traversed forward or backward in time, there are only three unique sequences. von Stryk and Glocker therefore evaluated all three sequences using a Nonlinear Programming (NLP) solver as the inner-loop to find the minimum-time trajectory of the salesman's car for each sequence. Chilan and Conway [45] solved a more challenging version of the MTSP where there are 8 cities and the salesman need visit only 3 of them. Since there are 168 unique sequences in the modified MTSP, it is time consuming to evaluate all of them. The amount of time necessary to evaluate every possible sequence of cities is equal to:

$$T_{evaluate-all-sequences} = T_{evaluate-one-sequence}\left(n_c^3 - 3n_c^2 + 2n_c\right) \tag{2.4}$$

Since the time necessary to evaluate every sequence varies with n_c^3, total enumeration quickly becomes impractical. Instead, Chilan and Conway used Branch and Bound (B&B) to choose candidate sequences of cities and an NLP solver to find the

minimum-time trajectory for each sequence. They found the optimal sequence while evaluating only 22 of the 168 possible sequences.

2.4 Hybrid Optimal Control in Spacecraft Trajectory Design

The interplanetary mission design problem may be formulated as a Hybrid Optimal Control Problem. A finite-state outer-loop solver may be used to choose a sequence of discrete mission events. A real-valued inner-loop solver is then required to evaluate a performance metric, such as total mission ΔV or mass remaining at the end of the mission, for each candidate event sequence. Several researchers have used this approach in the past. Wall and Conway optimized continuous-thrust missions to asteroids using an HOCP that used a binary Genetic Algorithm (GA) outer-loop solver to choose the sequence of asteroids and a real-valued GA combined with a novel shape-based trajectory approximation method to determine the trajectory between targets [44]. Chilan and Conway used an HOCP approach to solve the problem of rendezvous between two spacecraft that begin on opposite sides of a circular orbit and also to design a mission to Mars [41]. In both cases, a discrete outer-loop solver based on a binary GA was used to choose the sequence of propulsive and non-propulsive events (impulses, continuous-thrust arcs, and coast arcs), and an inner-loop solver using both a real-valued GA and an NLP solver was used to find the optimal trajectory.

However, none of the previous HOCP approaches to spacecraft trajectory design could be used to optimize missions using both propulsive maneuvers and planetary flybys. Flybys may be used to make significant changes to a spacecraft's velocity without expending propellant, and therefore may be used to reduce the cost of missions or even enable missions that were previously impossible. Therefore, a complete

interplanetary trajectory HOCP must be capable of designing missions that use flybys.

In this work, an HOCP solver is developed that can design the sequence of events and trajectory between pairs of user-defined end points. A key requirement and novel feature of this HOCP solver is that it works autonomously. That is, it needs only to be given very basic information regarding the desired mission, such as the allowed range of dates for the mission's beginning and end, and some engineering information about the spacecraft and launch vehicle.

In order to clearly define which events are part of the problem assumptions and which are chosen by the outer-loop solver, it is convenient to define three layers of event types: *missions*, *journeys*, and *phases*. A *mission* is a top-level container which encompasses all of the events, both user-defined and autonomously chosen. A mission is composed of *journeys*, each of which begins and ends at an object chosen by the user. For example, the interplanetary cruise portion of the Cassini mission was composed of a single journey that began at Earth and ended at Saturn. JAXA's Hayabusa mission, which rendezvoused and took samples from near Earth asteroid Itokawa, had two journeys - one from Earth to Itokawa, and one from Itokawa to Earth. NASA's Dawn mission is also composed of two journeys, one from Earth to Vesta and one from Vesta to Ceres. Each journey is composed of one or more *phases*. Like a journey, a phase begins at a planet and ends at a planet, but unlike the end points of a journey, the end points of a phase are chosen autonomously by the outer-loop solver. For example, the first journey of the Dawn mission (Earth to Vesta) is composed of two phases. The first phase begins with departure from Earth and ends at Mars, at the start of a Mars flyby. The second phase begins at the Mars flyby and ends at rendezvous with Vesta.

The outer-loop solver determines both the number of flybys to perform - and thus the number of phases - and also the planets at which those flybys will be performed.

Figure 2.2 is a block diagram of a mission using the above nomenclature. A real-valued inner-loop solver is then used to evaluate a cost function function value for each candidate flyby sequence. In this work, three different inner-loop trajectory models are presented, one for impulsive-thrust missions without deep-space maneuvers, one for impulsive-thrust missions with deep-space maneuvers, and one for continuous-thrust missions. A flow-chart of the outer-loop solution process is shown in figure 2.3.

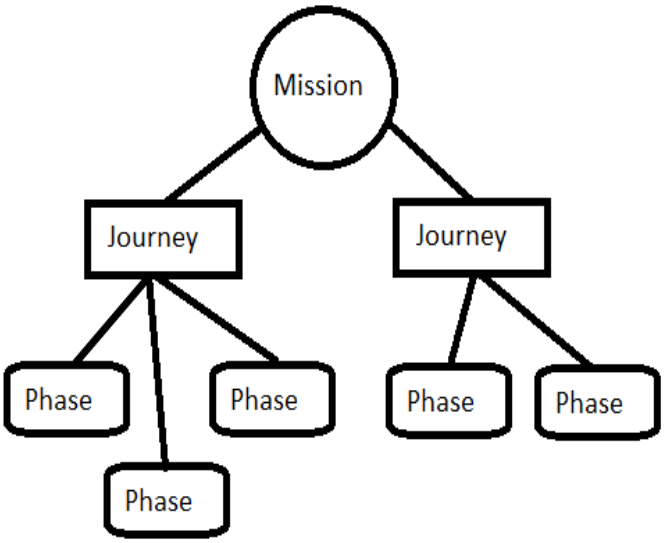


Figure 2.2: Anatomy of a Mission

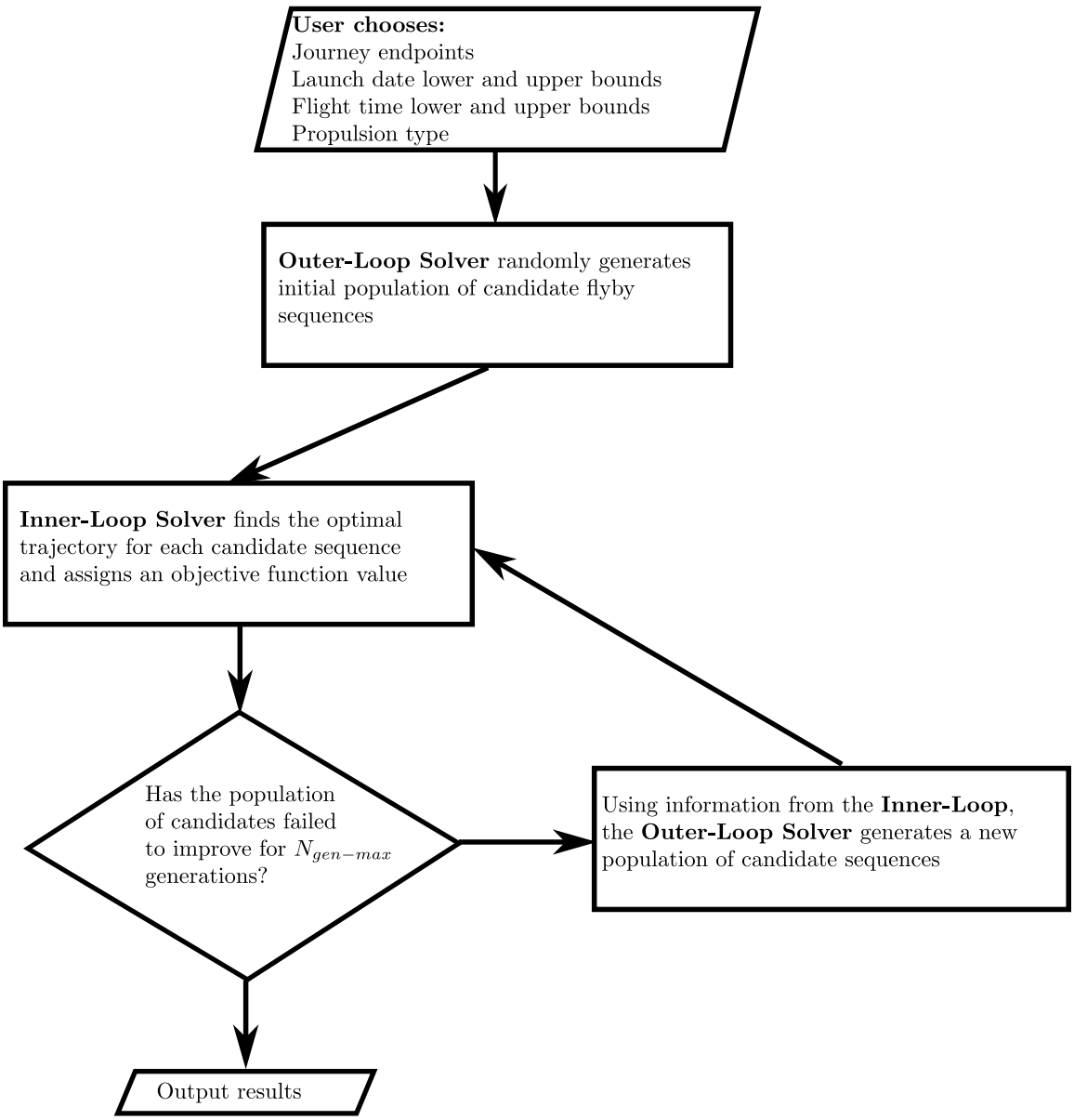


Figure 2.3: HOCP Solution Process

Chapter 3

Outer-Loop Optimization (of the Flyby Sequence)

3.1 Introduction

One of the most challenging aspects of interplanetary trajectory optimization is choosing the sequence of planetary flybys. This choice is especially difficult because one must not only choose the planets to be used for flybys, but also the number of flybys. For example, the Galileo mission performed a flyby of Venus and two flybys of the Earth on the way to Jupiter [8]. To design such a mission, one would first have to determine that there would be three flybys and only then could the targets of those flybys be chosen.

Previous researchers have addressed the problem of choosing the flyby sequence in several ways. A graphical tool, such as Pyxis [46] may be used to choose the flyby sequence by hand. Other researchers have used branch and bound techniques [39] or genetic algorithms [37, 38].

In this work, the interplanetary trajectory design problem is formulated as a hybrid optimal control problem (HOCP) with two nested optimization loops: an "outer-loop" that determines the sequence of flybys and an "inner-loop" that finds the optimal trajectory for each candidate flyby sequence. The cost function value of of the optimal solution to the inner-loop problem is then assigned as the cost function value of each sequence, allowing the outer-loop solver to compare different candidate sequences and choose the optimal sequence.

A necessary step in the creation of a HOCP mission design automaton is to choose

a suitable method to transcribe the flyby sequence, i.e. a way to represent it as a discrete optimization problem, and then to choose a suitable discrete optimization method. The approach here is to let each planet be identified by a unique integer and then the outer-loop problem can then be solved as an integer programming problem. A suitable integer programming problem solver must be chosen. Two such solvers are considered here: Branch and Bound (B&B) and the Genetic Algorithm (GA).

3.2 Branch and Bound

Branch and Bound (B&B) is a discrete optimization method that has been used in several HOCP formulations, including approaches to the Traveling Salesman Problem (TSP) and Motorized Traveling Salesman Problem (MTSP) [40, 41, 43], and the planning of spacecraft missions to asteroids [39]. In B&B, the set of all possible event sequences is represented as a tree. B&B then successively evaluates each branch of the tree. At each level of the tree, a partial sequence is evaluated and compared to the best known, or "incumbent," sequence. For a problem whose cost function obeys the principle of superposition, i.e.,

$$f(A+B) = f(A) + f(B) \tag{3.1}$$

where A and B are candidate events in the sequence, a partial event sequence is a relaxed subset of a full-length event sequence. In such cases, the cost function value of a partial event sequence is a lower bound on the cost function value of any full-length event sequence that includes that partial sequence. Therefore B&B immediately discards any partial sequence whose cost function value exceeds that of the incumbent full-length sequence, and also the entire branch of the decision tree to which that partial sequence belongs. This process continues until all branches have either been evaluated or discarded.

Figure 3.1 depicts a decision tree for a sequence of up to three events of types "A," "B," and "C." In this example, B&B traverses the tree starting at Level 0. The starting position is called the "parent" node. B&B then selects a "child" node from Level 1. The chosen "child" node becomes the first event of the sequence and also the "parent" for the next selection, this time from Level 2. The process continues until B&B reaches the bottom of the tree, in this case Level 3, or until the cost function value of the current partial sequences exceeds that of the incumbent.

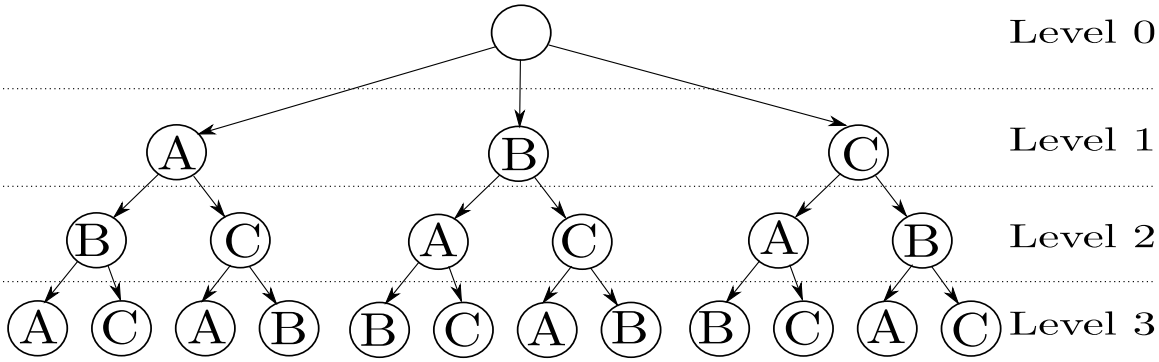


Figure 3.1: Example of Branch and Bound Decision Tree

Previous researchers who have used B&B successfully as an outer-loop solver for a HOCP have solved problems whose cost functions obey the principle of superposition. For example, the cost function value of traveling between a given pair of cities in the TSP is fixed and therefore the cost function value of any sequence of cities is equal to the sum of the cost function value of each pair of cities in the sequence. In the MTSP, there are many feasible paths between each pair of cities only one optimal path. Since the salesman must stop at each city, each leg of the trip is independent of those before and after it and therefore, as in the TSP, the cost function value of any sequence of cities is equal to the sum of the cost function value of each pair of cities in the sequence. Similarly, in the multiple-asteroid rendezvous problem [39], the spacecraft must match both position and velocity with each asteroid in the sequence, making it a variant of the MTSP where the "cities," in this case the asteroids, are moving. Because the spacecraft much match position and velocity with each target,

each phase of the mission is independent of the phase before and after just as in the TSP and the MTSP. Therefore the cost function value of a sequence of asteroid rendezvous is the sum of the cost function value of each rendezvous in the sequence.

The cost function of the outer-loop of the multiple-flyby interplanetary trajectory optimization does not obey the principle of superposition, i.e.,

$$f(A+B) \neq f(A) + f(B) \tag{3.2}$$

This is because the planetary flyby events are not only constraints but also may reduce the cost function value by providing a change in velocity without requiring that the spacecraft expend propellant. For example, the case of a trajectory from Earth to Saturn with a single Venus flyby (EVS) is not a relaxed subset of a trajectory from Earth to Saturn with flybys of Venus and Jupiter (EVJS) because the cost function value of EEJS may be less than that of EES. Suppose that the incumbent sequence is a direct Earth to Saturn (ES) trajectory, and then B&B evaluates the partial sequence EVS. If EVS has a greater cost function value than ES, then B&B will discard it and in doing so will also discard the entire branch that includes EVJS. However, in this example, EVJS has a less cost function value than both ES and EVS, so B&B will prune away the optimal solution. Therefore B&B is not an appropriate optimizer for the problems presented in this work.

3.3 Genetic Algorithm

The Genetic Algorithm (GA) is another global optimization solver which can be used to solve integer programming problems, one which does not suffer from the limitations of Branch and Bound. First proposed by Holland [47], the GA is a population-based search heuristic that mimics the evolution of a species. The GA has been shown to be effective as an outer-loop solver for hybrid optimal control problem spacecraft

trajectory optimization [29, 45] because the choice of a sequence of discrete events is analogous to the choice of a sequence of genes in the biological systems that the GA is designed to mimic.

The GA first generates a random population P of decision vectors, or *chromosomes*, in an N_p-dimensional vector space. The algorithm is run for a number of generations. At each generation, the fitness of every individual in the population is evaluated. A *selection* operator is employed to choose the best $N_{parents}$ individuals who will become the parents of the next generation. A *crossover* operator is then used to create child vectors from the chosen parent vectors, replacing a user-specified fraction of the population. This fraction is called the *crossover ratio*, or CR_{GA}. Then, with some small probability μ_{GA}, random *mutation* occurs in the resulting new population. Unfortunately this can sometimes result in the GA discarding the optimal solution after it is found. It is therefore desirable to preserve a small number N_{elite} individuals, termed *elite*, without modification in each generation. The process is then repeated either for a set number of generations N_{gen} or until some user-specified convergence criterion is met, such as the GA "stalling," that is, failing to improve the best known solution for $N_{stall-gen}$ generations. The basic GA is listed in Algorithm 3.1.

A GA can operate on strings of binary, integer, or real-valued variables. Because the flyby sequencing problem uses integer encoding, a binary or integer GA is most suitable. While one can use a publicly available GA such as Parallel Genetic Algorithm Package (PGAPack) [48], Genetic Algorithm Library (GALib) [49], or the Shark machine learning library [50], none of these packages were easily adaptable to the requirements of the HOCP automaton software architecture. The Multiple Gravity Assist (MGA) version of the HOCP automaton uses the MATLAB Global Optimization Toolbox [51, 52]. The Global Optimization Toolbox is designed to work with either binary or real-valued variables. In this case, the binary encoding was

Algorithm 3.1 Genetic Algorithm (GA)

generate P, a set of N_p random vectors $\mathbf{x}_{GA}$ in the decision space
evaluate all $\mathbf{x}_i \in P$
while not hit stop criterion **do**
  for i = 1 to $CR_{GA}N_p$ **do**
    choose $N_{parents}$ individuals from the population, forming a parent pool
    apply a *selection* operator to choose one vector $\mathbf{x_{mom}}$ from the parent pool
    choose $N_{parents}$ individuals from the population, forming a new parent pool
    apply a *selection* operator to choose one vector $\mathbf{x_{dad}}$ from the parent pool
    apply a *crossover* operator to generate $\mathbf{x_{child}}$ from $\mathbf{x_{mom}}$ and $\mathbf{x_{dad}}$
    insert $\mathbf{x_{child}}$ into the new population
  end for
  for i = 1 to $(1 - CR_{GA})\, N_p - N_{elite}$ **do**
    choose $N_{parents}$ individuals from the population, forming a pool
    apply a *selection* operator to choose one vector $\mathbf{x_{retained}}$ from the pool
    apply a *mutation* operator to $\mathbf{x_{retained}}$ to create $\mathbf{x_{mutated}}$
    insert $\mathbf{x_{mutated}}$ into the new population
  end for
  for i = 1 to N_{elite} **do**
    insert the i-th best individual $\mathbf{x_{elite-i}}$, unmodified, into the new population
  end for
  evaluate all $\mathbf{x}_i \in P$
end while
return $\mathbf{x}_{best}$

used, and the resulting binary values were used to form an integer-valued decision vector. The Multiple Gravity Assist with one Deep Space Maneuver (MGA-DSM) and the Multiple Gravity Assist with Low Thrust (MGA-LT) versions of the HOCP automaton are written entirely in C++ for better performance. Since none of the publicly available GA packages were easily integrated into the EMTG, an integer genetic algorithm was developed in C++ and used for this work. The selection operator is "tournament selection," where a "pool" of parents is selected from the population. The parent pool is then sorted and the best member of the pool is chosen to serve as a parent. This process is repeated every time a parent is required (twice per crossover operation). The well known binary crossover and uniform mutation operators are used [47]. The GA is run until either a set number of generations have occurred or the fitness has "stalled," i.e. not improved for a certain number of generations. Table 3.1 shows the settings used by the GA.

Table 3.1: Parameters for the Outer-Loop GA

Parameter	Description	Value
N_p	Population size	100
N_{gen}	Maximum number of generations	40
$N_{stall-gen}$	Maximum number of stall generations	5
CR_{GA}	Crossover ratio	0.3
$N_{parents}$	Size of parent pool	4
μ_{GA}	Mutation probability	0.05
N_{elite}	Elite count	1

An additional step is taken to improve the speed of the GA. Noting that each evaluation of an outer-loop fitness function requires solving an inner-loop optimization problem and that there is no need to optimize a mission sequence more than once, each solution is saved to a database after it is generated. Then if the outer-loop GA encounters the same mission sequence more than once, the fitness can simply be read from the database. Optimizing a mission sequence takes between one and several hundred seconds depending on the complexity of the mission, so replacing this with

a database lookup significantly reduces execution time.

3.4 Outer-Loop Transcription

The outer-loop problem is converted into an integer programming problem by representing each candidate flyby target with an integer code. The outer-loop GA can then choose a sequence of flybys by evolving an integer-valued decision vector. However, there is an additional difficulty to be overcome in that not all mission sequences will have the same number of flybys but the GA requires a decision vector of fixed length.

One existing method to circumvent this problem is the variable length genetic algorithm (VGA) developed by Kajitani *et al.* [53] to evolve combinatorial electronic circuits. Combinatorial circuit design requires choosing from a set of fuses that may be configured for positive input, negative input, or disconnected. Accordingly, each entry in the VGA chromosome contains two values - an address for a fuse and the value of that fuse, which may be positive or negative. If the chromosome does not contain the address of a certain fuse then that fuse is considered to be disconnected. This framework does not work well for the multiple gravity assist problem because unlike a fuse that may be used only once, a given planet may be "connected," that is used for a gravity assist, multiple times. The HGGA and DSMPGA methods developed by Gad and Abdelkhalik [37,38] are not suitable here because they are designed for a problem formulated as a single optimization loop, whereas for the HOCP formulation a method that solves only the sequencing problem, i.e. the outer-loop is required.

For this work we devised a method using "null" values. When the outer-loop fitness function parses a decision vector, it will skip over the null values and construct a trajectory only from the values that represent planets. This concept is similar to the "junk genes" in the field of genetics, where DNA chromosomes contain some genes that do not code for any traits and are skipped over when the chromosome is read [54].

This way the outer-loop optimization algorithm can not only evolve the sequence of flybys, but also the length of that sequence without having to specifically encode the length as a decision variable.

The numbers 0 to 15 are chosen to code for each body and the null body. The integer codes for each planet are contained in Table 3.2.

Table 3.2: Integer Codes for Destination Bodies

Integer Code	Body Name
0	Null
1	Mercury
2	Venus
3	Earth
4	Mars
5	Jupiter
6	Saturn
7	Uranus
8	Neptune
9-15	Null

For example, consider a mission that departs from Earth and ends at Jupiter. The outer-loop optimizer might generate the following integer string: $\begin{bmatrix} 2 & 3 & 11 & 3 \end{bmatrix}$

Using Table 3.2, the integers are converted into the sequence VEnE, where "V" represents a Venus flyby, "E" represents an Earth flyby, and "n" represents a null flyby that is not processed. The starting planet is prefixed to the sequence, and the destination is appended, yielding the sequence EVEEJ. This is a mission that departs from Earth, and then performs a gravity-assist maneuver at Venus and two at Earth before arriving at Jupiter. This is the sequence that was used by the Galileo mission [8].

Chapter 4

Inner-Loop Optimization (of Impulsive-Thrust Trajectories)

4.1 Introduction

Most interplanetary missions use chemical rocket motors, which are characterized by a high thrust and a relatively low specific impulse (I_{sp}). The high thrust means that a chemical motor can significantly change the velocity of the spacecraft in a short time, and the low I_{sp} means that the motor burns propellant quickly and so cannot be operated for more than a few minutes. As a result, spacecraft that use chemical propulsion tend to perform short but powerful rocket burns that last seconds or minutes, and then coast for weeks, months, or even years without maneuvering. As a result, two simplifying assumptions may be made that make designing chemical propulsion missions much simpler.

First, the time spent thrusting is a very small fraction of the total flight time. It is therefore reasonable to model a chemical rocket burn as an instantaneous, or impulsive, change in velocity (Δv) [55]. Second, because the spacecraft coasts for long periods of time without performing maneuvers, it is reasonable to assume that the motion of the spacecraft follows Kepler's laws. This assumption neglects perturbation forces that may act upon the spacecraft, such as the gravity of distant planets or solar radiation pressure, but is a reasonable assumption for preliminary mission design because the perturbing forces are very small. Because of the assumption of Keplerian orbits, direct numerical integration of the spacecraft trajectory may be avoided. Instead, the position and velocity of the spacecraft may be propagated analytically by

solving Kepler's problem [55]. Also, there exists a closed form solution to the problem of finding a conic section arc between two points in space. If two points in space, such as the positions of Earth and Mars at an initial and final epoch, and the flight time are known, Lambert's method may be used to find a trajectory that connects them. These simplifying assumptions allow very complex problems to be reduced to a form which is computationally inexpensive so that they may be solved in a HOCP framework.

4.1.1 Multiple Gravity Assist (MGA)

The first trajectory model presented here is the Multiple Gravity Assist (MGA) model described by Vink and Izzo [15]. A set of simplifying assumptions are made. First, all rocket burns are assumed to be impulsive. Second, impulses are allowed only at launch and at the moment of periapse passage in each gravity-assist maneuver. Finally, a two-body, patched-conic model is used, allowing the path of the spacecraft between two bodies to be modeled using the Lambert solution to the problem of finding the conic section orbit between two points in specified time [55]. The specific Lambert solver used was developed by Izzo *et al.* for the Global Trajectory Optimization Problem (GTOP) database [56].The resulting Lambert arcs are patched at each intermediate destination. Since both Lambert arcs are guaranteed to touch one endpoint to the position of the planet, we must only account for the difference in velocity between the end point of the incoming arc and the starting point of the outgoing arc. This velocity discrepancy is resolved by modeling the flyby about the planet.

Each flyby is considered to take place at the location of the planet in the heliocentric reference frame. By solving Lambert's problem, we can determine the spacecraft's incoming and outgoing velocity vectors in the heliocentric frame for each flyby. Then

we can find the incoming and outgoing velocities relative to the planet as:

$$\mathbf{v}_{\infty/in} = \mathbf{v}_{sc/in} - \mathbf{v}_{pl} \tag{4.1}$$

$$\mathbf{v}_{\infty/out} = \mathbf{v}_{sc/out} - \mathbf{v}_{pl} \tag{4.2}$$

These velocity vectors are the asymptotes of the incoming and outgoing hyperbolic orbits about the planet, respectively. The flyby transfer angle δ is thus:

$$\delta = \left\langle \mathbf{v}_{\infty/in}, \mathbf{v}_{\infty/out} \right\rangle \tag{4.3}$$

where the brackets indicate the angle between two vectors. One can then solve iteratively for the flyby periapse distance r_p by setting:

$$\sin^{-1}\left(1/e_{in}\right) + \sin^{-1}\left(1/e_{out}\right) = \delta \tag{4.4}$$

where

$$e_{in} = 1 + \frac{r_p v_{\infty/in}^2}{G m_{pl}} \tag{4.5}$$

and

$$e_{out} = 1 + \frac{r_p v_{\infty/out}^2}{G m_{pl}} \tag{4.6}$$

Once r_p is known, the magnitude of the rocket burn at periapse can be found by:

$$\Delta v = \left| \sqrt{v_{\infty/in}^2 + \frac{2Gm_{pl}}{r_p}} - \sqrt{v_{\infty/out}^2 + \frac{2Gm_{pl}}{r_p}} \right| \tag{4.7}$$

since the Δv at periapse is parallel to the instantaneous velocity. Thus the flyby periapse distance and the magnitude of the rocket burn for each flyby are found directly from the spacecraft's incoming and outgoing velocities at the planet, which in turn are found by solving Lambert's problem. Figure 4.1 illustrates the flyby.

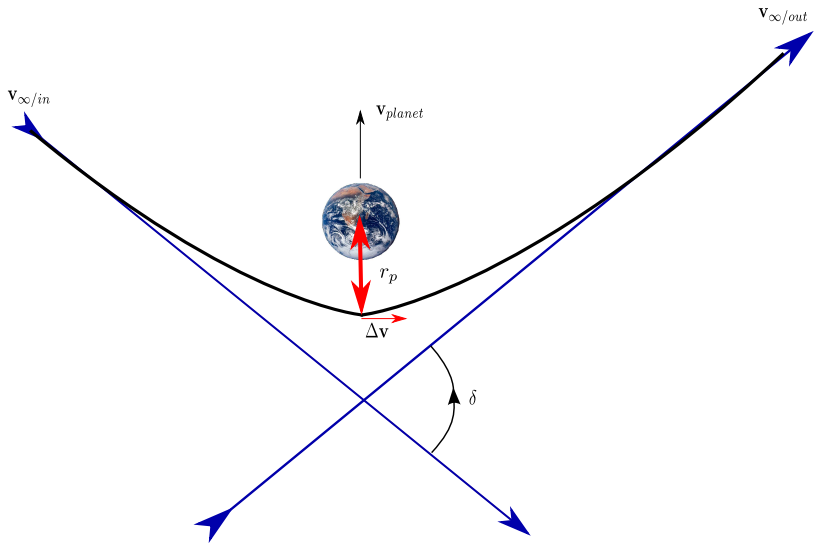


Figure 4.1: MGA Flyby Geometry

The MGA model reduces the aforementioned complex optimization problem to:

$$\begin{aligned} &\text{Minimize } f(\mathbf{x}) + g(\mathbf{x}) \\ &\text{where:} \\ &\mathbf{x} = \begin{bmatrix} T_{launch} & T_1 & T_2 & \dots & T_{n-1} \end{bmatrix} \end{aligned} \tag{4.8}$$

Here T_{launch} is the date of launch, T_i is the flight time from the ith planet in the sequence to the *(i+1)*th planet, and $f(\mathbf{x})$ can be any relevant fitness function, typically the sum of the magnitudes of all of the impulses, and $g(\mathbf{x})$ is a penalty function enforcing mission constraints that will be described in Section 4.1.3. The value n refers to the number of planets, so T_{n-1} refers to the flight time from the second to last planet to the last planet. The resulting fitness function allows the inner-loop problem to be optimized without explicit path constraints, making it ideal for use

with evolutionary algorithms. A diagram of a simple two-phase Earth-Venus-Mars (EVM) MGA mission is shown in Figure 4.2.

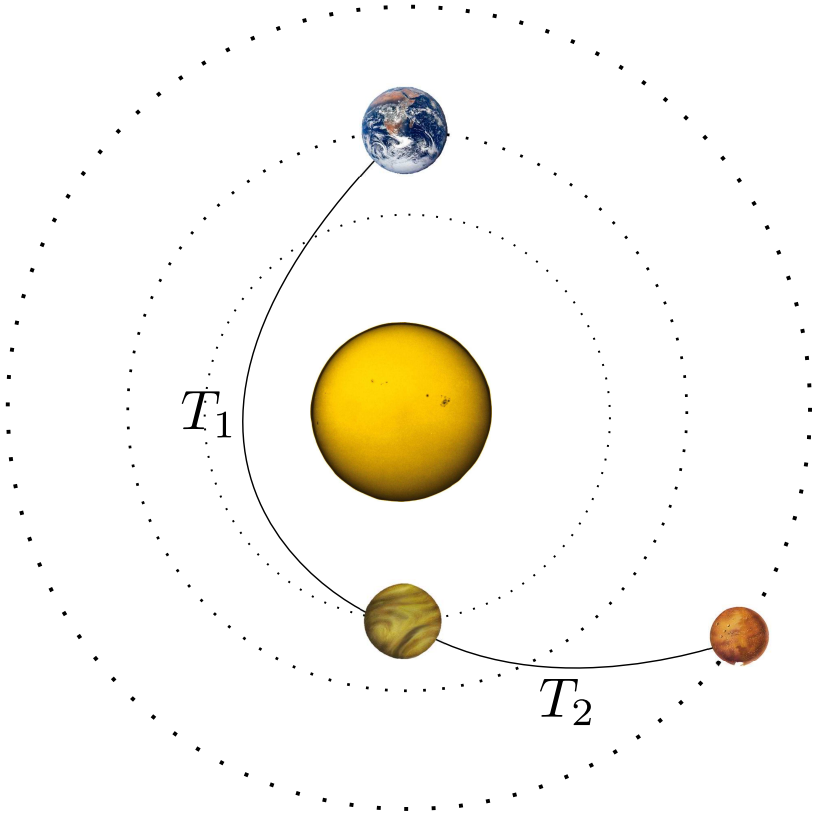


Figure 4.2: Diagram of a Two-Phase MGA Mission

In the MGA case, it is possible that a powered flyby may require the spacecraft to pass through the planet in order to achieve the desired flyby transfer angle. This is clearly not feasible, so a penalty is imposed on this behavior so that the evolutionary algorithm will evolve away from solutions with this problem. The penalty is formulated as

$$g\left(\mathbf{x}\right) = \sum\nolimits_{i=1}^{n-1} g_i\left(\mathbf{x}\right) \tag{4.9}$$

where

$$g_i(\mathbf{x}) = -2\log_{10}\left(\frac{r_{p,i}}{1.05 r_{pl,i}}\right) \tag{4.10}$$

Thus if the required flyby periapse radius is less than 1.05 times the radius of the planet, a logarithmically scaled penalty is applied. The value of 1.05 is chosen to allow the spacecraft to fly close to the planet, i.e. at an altitude of 5% of the planet's radius, but not so close that it enters the planet's atmosphere.

4.1.2 Multiple Gravity Assist with Deep Space Maneuver (MGA-DSM)

While simple to formulate and comparatively straightforward to optimize due to the small number of decision parameters, the MGA model is not suitable for all missions because the constraint that rocket burns occur only at the periapse of planetary flybys is quite restrictive. For example, the Cassini mission required a large deep space maneuver [10]. The optimal location for a rocket burn may not be the periapse of a flyby, and a lower cost may be found by relaxing this constraint. For this reason, the Multiple Gravity-Assist with one Deep Space Maneuver (MGA-DSM) model was developed, first by Vasile and De Pascale [14], and then by Vinkó and Izzo [15]. In the MGA-DSM model a single impulse is allowed at any point along each planet-to-planet phase. No impulse is permitted at periapse of the flybys.

In the first phase of an MGA-DSM model, the spacecraft is assumed to start with the same state vector as the planet it launches from. An impulse, with magnitude bounded by the maximum heliocentric velocity that the launch vehicle can impart to the spacecraft, is applied in a chosen direction. This impulse is defined by three decision variables, a magnitude Δv_{LV} and two angles RLA and DLA that describe the right ascension and declination of the launch asymptote. Two additional decision

variables, T_1 and η_1, encode the flight time for the first phase and the burn index, respectively. The burn index is a number in the range $[0, 1]$ that defines at what point along the spacecraft's arc the burn occurs. The spacecraft's initial state is defined as

$$\mathbf{r}_{s/c}\left(T_{launch}\right) = \mathbf{r}_{pl}\left(T_{launch}\right) \tag{4.11}$$

$$\mathbf{v}_{s/c}\left(T_{launch}\right) = \mathbf{v}_{pl}\left(T_{launch}\right) + \Delta v_{LV} \begin{bmatrix} \cos RLA \cos DLA \\ \sin RLA \cos DLA \\ \sin DLA \end{bmatrix} \tag{4.12}$$

The spacecraft's state vector is then propagated forward by $\eta_1 T_1$ days and then the deep space maneuver impulse is applied. The second half of the phase is modeled as a Lambert arc. The spacecraft must arrive at the next planet in the sequence at the end of that arc. The desired final position vector may be found by locating the planet at time $t = T_{launch} + T_1$. The flight time is simply $(1 - \eta_1) T_1$. Lambert's problem is solved to find the transfer arc, and the required deep space maneuver Δv_1 is just the magnitude of the vector difference between the spacecraft's velocity before and after entering the Lambert arc.

For every phase after the first, the process is very similar except that instead of starting with a propulsive maneuver, the remaining phases start with a flyby. The flyby is parameterized by two decision variables, the altitude ratio R_p and the b-plane insertion angle γ. The altitude ratio defines the periapse distance of the flyby r_p by

$$r_p = R_p r_{pl} \tag{4.13}$$

where r_{pl} is the radius of the planet. Then, knowing the incoming heliocentric velocity

of the spacecraft, which is the same as its velocity at the end of the previous phase, and the mass of the planet, we can compute the velocity after the flyby. Once again we compute the incoming velocity of the spacecraft relative to the planet:

$$\mathbf{v}_{\infty/in} = \mathbf{v}_{sc/in} - \mathbf{v}_{pl} \tag{4.14}$$

Because this flyby is unpowered, we assume that the spacecraft will follow a hyperbolic path about the planet and therefore the magnitudes of the incoming and outgoing velocity vectors relative to the planet are equal, i.e.

$$v_{\infty/out} = v_{\infty/in} \tag{4.15}$$

The algorithm must then find the orientation of the outgoing velocity vector. First, the eccentricity of the hyperbola is found by

$$e = 1 + \frac{r_p \mathbf{v}_\infty^2}{G m_{pl}} \tag{4.16}$$

and then the flyby transfer angle (shown in Figure 4.1) may be found by

$$\delta = 2 \sin^{-1}(1/e) \tag{4.17}$$

Finally, the incoming velocity vector must be rotated about the axis normal to the flyby hyperbola. The outgoing velocity vector is

$$\mathbf{v}_{\infty/out} = v_{\infty/in} \left(\cos\delta \hat{i} + \cos\gamma \sin\delta \hat{j} + \sin\gamma \sin\delta \hat{k} \right) \tag{4.18}$$

where $\hat{i}$, $\hat{j}$, $\hat{k}$ are the unit vectors defining the b-plane:

$$\hat{i} = \frac{\mathbf{v}_{\infty/in}}{v_{\infty/in}} \tag{4.19}$$

$$\hat{j} = \frac{\hat{i} \times \mathbf{v}_{pl}}{\left\| \hat{i} \times \mathbf{v}_{pl} \right\|} \tag{4.20}$$

$$\hat{k} = \hat{i} \times \hat{j} \tag{4.21}$$

The outgoing heliocentric velocity vector can then be found as

$$\mathbf{v}_{sc/out} = \mathbf{v}_{\infty/out} + \mathbf{v}_{pl} \tag{4.22}$$

As in the first phase of the mission, the spacecraft motion is then propagated in time by $\eta_i T_i$, after which time Lambert's problem is solved again to find the arc and associated deep space maneuver necessary to bring the spacecraft to the next planet in the sequence. If this planet is the last in the sequence, an intercept (flyby), rendezvous, or orbit insertion may be performed depending on the mission requirements.

The MGA-DSM problem may then be summarized as

$$\begin{gathered} \text{Minimize } f(\mathbf{x}) + g(\mathbf{x}) \\ \text{where :} \\ \mathbf{x} = \begin{matrix} [T_{launch} & \Delta v_{LV} & RLA & DLA & T_1 & \dots & T_{n-1} & \eta_1 & \dots & \eta_{n-1} \\ R_{p2} & \dots & R_{p(n-1)} & \gamma_2 & \dots & \gamma_{n-1}] \end{matrix} \end{gathered} \tag{4.23}$$

As in the MGA case, $f(\mathbf{x})$ can be any relevant fitness function, but is typically the sum of the magnitudes of all of the impulses, and $g(\mathbf{x})$ is a penalty function

enforcing mission constraints. For example, one might wish to constrain flight time or arrival date by letting $g(\mathbf{x})$ equal the amount by which the flight time or arrival date exceed a chosen value. Additional constraints are described in Section 4.1.3. A simple two-phase MGA-DSM mission is shown in Figure 4.3.

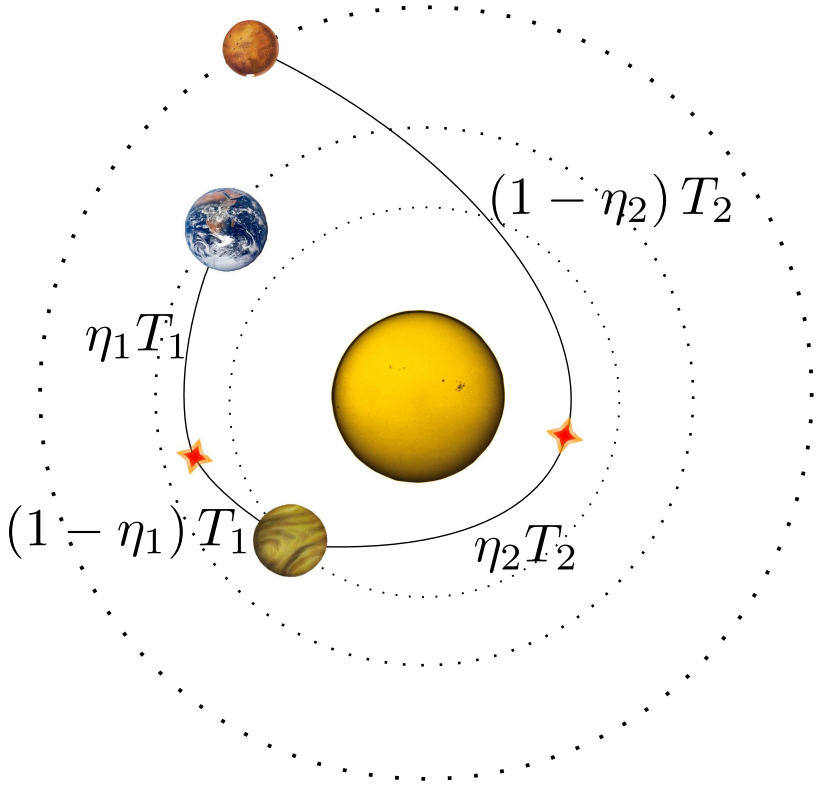


Figure 4.3: Diagram of a Two-Phase MGA-DSM Mission

4.1.3 Additional Constraints

The mission designer may wish to impose any of a number of different types of terminal or path constraints. Such constraints are used to model arrivals and departures, and also to ensure that all flybys are physically possible (i.e. do not hit the planet or violate the laws of physics). Most importantly, the user must choose the type of arrival maneuver for each journey. Possible choices include a rendezvous (match speed and velocity), an intercept (match position only) with bounded incoming velocity, or insertion into an orbit about the target via a chemical burn. The latter reduces

the final mass of the spacecraft by using propellant necessary to provide the velocity change:

$$\Delta v_{insertion} = \left| v_{p/in} - v_{p/desired} \right| \tag{4.24}$$

Here $v_{p/in}$ is the velocity of the spacecraft at periapse of the arrival hyperbola,

$$v_{p/in} = \sqrt{v_\infty^2 + 2\mu_{planet}/r_p} \tag{4.25}$$

where v_∞ is the incoming velocity asymptote, μ_{planet} is the gravitational parameter of the planet, and r_p is both the distance from the center of the planet at periapse of the incoming hyperbola and also the periapse of the desired orbit. The velocity necessary to be in the desired orbit, $v_{p/desired}$, is given by:

$$v_{p/desired} = \sqrt{\mu_{planet}\left(2/r_p - \frac{1}{a_{desired}}\right)} \tag{4.26}$$

where $a_{desired}$ is the semi-major axis of the desired orbit.

An additional constraint is required for MGA and MGA-DSM problems. Both models require that the spacecraft follow a hyperbolic path about the planet during the flyby. If the spacecraft's incoming velocity is too low, then the planet-centered orbit is not a hyperbola and the model is invalid. This can happen when the spacecraft launches and then performs a flyby of the same planet it launched from. Such repeated flybys are not uncommon; for example, the Rosetta mission performed a flyby of Earth one year after launch without visiting any other planets between launch and the Earth encounter [12]. It was discovered that sometimes the inner-loop solver chooses a very low launch impulse, so the spacecraft never actually leaves the starting planet. When it comes time to compute the flyby, unpredictable non-physical behavior is observed.

This problem was found to be related to the assumption that during a flyby, the trajectory of the spacecraft is hyperbolic relative to the planet. If the spacecraft's velocity relative to the planet is too low, as in the case where the spacecraft never left the starting planet, then the actual orbit is not hyperbolic and the flyby equations are not valid. A solution to this problem is to constrain the energy of the spacecraft's orbit about the planet at the boundary of the planet's gravitational sphere of influence. The gravitational sphere of influence of the planet is defined [55] as:

$$r_{SOI} \approx \left(\frac{m_{pl}}{m_{\odot}}\right)^{2/5} r_{\odot pl} \tag{4.27}$$

Then the orbital energy of the spacecraft at the boundary of the sphere of influence where it enters orbit about the planet, is

$$E = \frac{v_{\infty/in}^2}{2}^2 - \frac{Gm_{pl}}{r_{SOI}} \tag{4.28}$$

For a hyperbolic orbit, E must be positive. Several penalty methods were considered to enforce a positive E. Since there is no need for a penalty when E is indeed hyperbolic, the penalty function $g(\mathbf{x})$ is defined piecewise - it is zero when E is positive and positive when E is negative or zero. Several candidates for $g(\mathbf{x})$ were considered. The first attempt was to let $g(\mathbf{x})$ equal the negative of E, i.e.,

$$g(\mathbf{x}) = \begin{cases} 0 & for \quad E > 0 \\ -E & for \quad E \leq 0 \end{cases} \tag{4.29}$$

The penalty method described in Equation 4.29 did not work well. As E approaches zero, $g(\mathbf{x})$ also approaches zero and so the inner-loop optimizer often found solutions that minimized the objective function $f(\mathbf{x})$ but left a small nonzero com-

ponent in $g(\mathbf{x})$. However even a small nonzero value of $g(\mathbf{x})$ represents an infeasible flyby, so this did not fix the problem. Next, a scaling factor k_{flyby} was added to $g(\mathbf{x})$, i.e.,

$$g(\mathbf{x}) = \begin{cases} 0 & for \quad E > 0 \\ -k_{flyby}E & for \quad E \leq 0 \end{cases} \tag{4.30}$$

Unfortunately the penalty method of Equation 4.30 did not work either. The scaling factor k_{flyby} only changed the threshold at which the optimizer ignored the constraint but did not actually enforce it. In addition, it was difficult to choose a suitable value for k_{flyby} *a priori*. A new approach was therefore tried; instead of using E directly in the penalty function, $v_{\infty/in}$, the incoming velocity relative to the planet, was used instead. It was necessary to find a function that would heavily penalize values of $v_{\infty/in}$ that were too low, i.e, that represented a non-hyperbolic approach to the planet. The first such method tried was to use the order of magnitude of $v_{\infty/in}$, i.e.,

$$g(\mathbf{x}) = \begin{cases} 0 & for \quad E > 0 \\ -\log_{10}\left(v_{\infty/in}\right) & for \quad E \leq 0 \end{cases} \tag{4.31}$$

The logarithmic penalty method was much more effective than the previous methods but still not ideal because $g(\mathbf{x})$ was so large for infeasible flybys that it dominated $f(\mathbf{x})$ and the optimizer was sometimes unable to find a solution. The next step was to find a function of $v_{\infty/in}$ that penalized low values but would not grow so large as the method of Equation 4.31. The inverse of $v_{\infty/in}$ was used, i.e.,

$$g(\mathbf{x}) = \begin{cases} 0 & for \quad E > 0 \\ \frac{1}{v_{\infty/in}} & for \quad E \leq 0 \end{cases} \tag{4.32}$$

This last penalty method was found to be very effective at penalizing non-hyperbolic flybys. However one concern remained. Equation 4.27 is only an approximation, as the actual radius of a planet's sphere of influence changes as the planet moves closer to and farther from the sun along its orbit. In fact, the "sphere of influence" is not even a true sphere. Modeling this properly would be challenging and computationally expensive, so a 10% margin was applied to $v_{\infty/in}$ to ensure that the penalty function worked as designed. The final form of the penalty function is thus:

$$g(\mathbf{x}) = \begin{cases} 0 & for \quad \frac{\left(0.9 v_{\infty/in}\right)^2}{2} - \frac{G m_{pl}}{r_{SOI}} \geq 0 \\ \frac{1}{v_{\infty/in}} & for \quad \frac{\left(0.9 v_{\infty/in}\right)^2}{2} - \frac{G m_{pl}}{r_{SOI}} < 0 \end{cases} \tag{4.33}$$

Equation 4.33 effectively prevents the non-hyperbolic flyby problem.

In addition, constraints may be imposed on the total mission flight time and/or the flight time or arrival date of each journey. For MGA and MGA-DSM problems, this constraint is formulated as a linear penalty function:

$$g(\mathbf{x}) = \begin{cases} 0 & for \quad \sum_{i=1}^{n-1} T_i \leq T_{flight-max} \\ k_{time}\left(\sum_{i=1}^{n-1}(T_i) - T_{flight-max}\right) & for \quad \sum_{i=1}^{n-1} T_i > T_{flight-max} \end{cases} \tag{4.34}$$

Here k_time is a scaling constant. The inner-loop optimizer would then vary $\mathbf{x}$ until the total time of flight is less than some user-specified $T_{flight-max}$, which in turn will drive $g(\mathbf{x})$ to zero. For the MGA and MGA-DSM examples presented in this paper, $k_{time} = 10$ has proven to be a very satisfactory choice.

4.2 Optimization Methods for Impulsive-Thrust Problems

Once a candidate flyby sequence has been chosen by the outer-loop solver, the inner-loop solver must find the optimal trajectory for that candidate sequence. That is, a trajectory must be found that optimizes a cost function of interest, such as minimum Δv or maximum mass delivered to a target, while visiting the set of planets chosen by the outer-loop. Impulsive-thrust trajectories are parameterized by a small number of decision variables and need only satisfy a small number of constraints, such as the flight time bounds and flyby feasibility constraints defined in Section 4.1.3, which can be formulated as penalty functions. Impulsive-thrust MGA and MGA-DSM problems are thus well suited to evolutionary algorithms (EAs).

Four different types of population-based evolutionary algorithms (EAs) are used in this work to solve MGA and MGA-DSM problems: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE), and a cooperative algorithm of PSO and DE (COOP). All four work by first generating a random "population" of solutions, and then, using stochastic operators, combine and modify that initial population to attempt to improve some cost function. EAs have two significant advantages: they do not require an initial guess of the solution and they often, although not always, can escape locally optimal solutions and find the global optimum. The latter is a major advantage for problems such as trajectory optimization that often have many local minima.

4.2.1 Real Genetic Algorithm

The first optimization method considered for impulsive-thrust problems is the real-valued genetic algorithm (GA). The real-valued GA is very similar to the binary-valued and integer-valued genetic algorithms used by the outer-loop solver described

in Section 3.3, but operates on chromosomes composed of real numbers instead of binary or integer values. In contrast to the outer-loop solver, the inner-loop GA operates on a fixed-length chromosome because the number of variables defining an MGA or MGA-DSM problem with a given number of flybys is fixed. Previous researchers have made extensive use of real-valued GAs to optimize impulsive-thrust missions [14, 15, 29, 37, 38].

However, in recent years the GA has been shown to be just one of many EAs that are suitable for the solution of MGA and MGA-DSM problems. In fact, the GA has been shown to be one of the less effective choices of solver [15]. The GA was briefly considered in this work as an inner-loop solver for MGA problems. In this case, the GA contained in MATLAB's Global Optimization Toolbox [51, 52] was used. After a few tests, it was found that the GA was less effective than Differential Evolution (DE) and Particle Swarm Optimization (PSO). The real-valued GA was therefore discarded as a solution method for MGA problems. This finding was in agreement with that of Vinkó and Izzo [15]. Since Vinkó and Izzo found that the real-valued GA was even less effective for MGA-DSM problems than for MGA, the GA was never applied to MGA-DSM problems in this work.

4.2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO), developed by Kennedy and Eberhart [57], is a population based evolutionary algorithm that approximates the behavior of groups of social animals, such as swarms of insects or schools of fish. Consider an n-dimensional vector space, and generate a population $\mathbf{P}$ of position vectors $\mathbf{u}_i$ within that space. Then, drawing from a uniform distribution, generate a random velocity vector ν_i for each individual in the population. Each individual can be considered a particle that moves freely in the space. At each iteration, the fitness of each particle is evaluated. The position vector corresponding to the best fitness value found by each individual

particle, $\mathbf{u}_{local-best}$, and the position vector corresponding to the best fitness value found by the entire swarm, $\mathbf{u}_{global-best}$ are computed and stored at each iteration. The system of particles is propagated through a series of iterations. The velocity and position vectors evolve at each iteration according to:

$$\begin{aligned}\nu(i,k) = w_1\nu(i,k-1) + w_2 \cdot rand[0,1] \cdot (\mathbf{u}_{local-best} - \mathbf{u}(i,k-1)) \\ + w_3 \cdot rand[0,1] \cdot (\mathbf{u}_{global-best} - \mathbf{u}(i,k-1))\end{aligned} \tag{4.35}$$

$$\mathbf{u}(i,k) = \mathbf{u}(i,k-1) + \nu(i,k) \tag{4.36}$$

Here $\mathbf{u}(i,k)$ is the position of particle i at iteration k, $\mathbf{q}(i,k)$ is the velocity of that particle, and the w_i are random weighting coefficients in a range chosen by the user.

The PSO solver must be forced to obey the upper and lower bounds on the decision variables otherwise the solutions are not useful. Therefore any particle that hits the user-imposed boundary of the decision space is given zero velocity in the direction that it contacted the boundary, and held at the boundary for that iteration. Finally, it is possible for the velocity magnitudes q_i to grow large enough to prevent the PSO from effectively exploring the decision space. To prevent this problem, the magnitude of the velocity of an individual particle is constrained to remain below some user-specified ν_{max}. The PSO algorithm is run until either a specified number of iterations is reached or the best solution fails to improve for a certain number of iterations. The PSO algorithm is listed as Algorithm 4.1.

4.2.3 Differential Evolution

Differential Evolution (DE), first developed by Storn and Price [58,59], is a population-based evolutionary algorithm that simulates biological evolution on real-valued decision vectors. There are many "strategies" of DE, and the specific one used here, as

Algorithm 4.1 Particle Swarm Optimization (PSO)

generate $\mathbf{P}$, a set of random position vectors $\mathbf{u}_i$ in the decision space
generate $\mathbf{P}_\nu$ a set of random velocity vectors ν_i in the decision space
while not hit stop criterion **do**
  for each position vector in P **do**
    evaluate $f(\mathbf{u}_i)$
    if $f(\mathbf{u}_i) < f(\mathbf{u}_{i-local-best})$ **then**
      $\mathbf{u}_{i-local-best} = f(\mathbf{u}_i)$
    end if
    if $f(\mathbf{u}_i) < f(\mathbf{u}_{global-best})$ **then**
      $\mathbf{u}_{global-best} = f(\mathbf{u}_i)$
    end if
    $\nu(i,k) = w_1\nu(i,k-1) + w_2 \cdot rand[0,1] \cdot (\mathbf{u}_{local-best} - \mathbf{u}(i,k-1))$
    $+w_3 \cdot rand[0,1] \cdot (\mathbf{u}_{global-best} - \mathbf{u}(i,k-1))$
    $\mathbf{u}(i,k) = \mathbf{u}(i,k-1) + \nu(i,k)$
    if $\|\nu_i\| > \nu_{max}$ **then**
      $\nu = \nu/\nu_{max}$
    end if
    for $k \in [1,n]$ **do**
      if $u_{i,k} > 1.0$ **then**
        $u_{i,k} = 1.0$
        $\nu_{i,k} = 0.0$
      elseif $u_{i,k} < 0.0$
        $u_{i,k} = 0.0$
        $\nu_{i,k} = 0.0$
      end if
    end for
  end for
end while
return $\mathbf{u}_{global-best}$

recommended by Vinkó and Izzo [15], is called *best/2/bin*. First, a trial population $\mathbf{P}$ is created of random vectors $\mathbf{u}_i$ in the decision space. Then, for each vector $\mathbf{u}_*$ in the population, a "difference vector" $\mathbf{d}$ is produced by:

$$\mathbf{d} = \mathbf{u}_1 - \mathbf{u}_2 + \mathbf{u}_3 - \mathbf{u}_4 \tag{4.37}$$

where $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, and $\mathbf{u}_4$ are randomly selected vectors from the population. Next, a binary crossover operation is performed to create a new "trial vector" $\mathbf{u}_{trial}$. For

each element i of $\mathbf{u}_*$, a random value ω is generated. If ω is less than some threshold "crossover ratio" CR, then:

$$u_{trial,i} = u_{best,i} + Fd_i \tag{4.38}$$

where $\mathbf{u}_{best}$ is the current best vector in the population and F is a scaling factor. Otherwise,

$$u_{trial,i} = u_{*,i} \tag{4.39}$$

The resulting $\mathbf{u}_{trial}$ is then evaluated, and its objective function value is compared to that of $\mathbf{u}_*$ and the better of the two is retained in the population. The "losing" vector is discarded.

Like PSO, DE does not naturally handle bounds on the decision variables, *i.e.* it is possible for a trial vector to be created that does not fall within the bounds. In this work, any element of $\mathbf{u}_{trial}$ that falls outside the bounds is replaced by a random value from within the bounds. DE is run until either a specified number of iterations is reached or the best solution fails to improve for a certain number of iterations. The *best/2/bin* differential evolution algorithm used in this work is listed as Algorithm 4.2.

Here n is the dimension of the decision space, $F \in [0, 1]$ is a scaling factor, and $CR \in [0, 1]$ is the crossover ratio, both of which are set by the user.

4.2.4 Cooperative Algorithm of DE and PSO

Unfortunately neither particle swarm nor differential evolution is sufficient to solve the MGA inner-loop problem alone. It was found that both algorithms tend to prematurely converge to local minima and a large number of runs are necessary to increase the probability of finding a good solution. Vinkó and Izzo [15] observed the

Algorithm 4.2 Differential Evolution (DE)

generate P, a set of random position vectors $\mathbf{u}_i$ in the decision space
evaluate all $\mathbf{u}_i \in P$ and generate u_{best}
while not hit stop criterion **do**
  for each vector $\mathbf{u}_* \in P$ **do**
    randomly choose vectors $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, $\mathbf{u}_4$ from P
    let $\mathbf{d} = \mathbf{u}_1 - \mathbf{u}_2 + \mathbf{u}_3 - \mathbf{u}_4$
    for $i \in [1, n]$ **do**
      generate $\omega \in random\,[0, 1]$
      if $\omega < CR$ **then**
        $u_{trial,i} = u_{best,i} + Fd_i$
      else
        $u_{trial,i} = u_{*,i}$
      end if
      if $u_{trial,i} < x_{lb,i}$ **or** $u_{trial,i} > x_{ub,i}$ **then**
        $u_{trial,i} = random\,[0, 1]$
      end if
    end for
    if $f(\mathbf{u}_{trial}) < f(\mathbf{u}_*)$ **then**
      $\mathbf{u}_* = \mathbf{u}_{trial}$
      if $f(\mathbf{u}_{trial}) < f(\mathbf{u}_{best})$ **then**
        $\mathbf{u}_{best} = \mathbf{u}_{trial}$
      end if
    end if
  end for
end while
return $\mathbf{u}_{best}$

same problem and speculated that the tendency to converge to local minima in the MGA problem was related to the dynamics, i.e. governing equations, of the DE and PSO solvers. Changing the solver dynamics might allow the solver to escape from a local minimum. Vinkó and Izzo proposed a "cooperative algorithm" that switches rapidly back and forth between DE and PSO, running each for a small number of iterations and then passing the population to the other. The velocity vectors for PSO are randomly generated for each run. This new algorithm, known as COOP, was found to be very effective for solving an Earth to Saturn benchmark problem [15]. COOP was also found to be a suitable inner-loop solver for MGA problems in this work.

The particle swarm parameters w_1, w_2, and w_3 are 0.65, 2.0, and 2.0 respectively, and the maximum velocity ν_{max} is 0.5. The differential evolution parameters F and CR are 0.8 and 0.9 respectively. For consistency, these are the same values used by Vinkó and Izzo. Because the population size of the optimizer must scale with the size of the problem being solved, a population size of $20n$ individuals is used, where n is the cardinality of the decision space. Each short PSO or DE run lasted for $10n$ generations. These values were found to be effective after several trials.

Two additional considerations must be addressed in designing the inner-loop solver. First, not all inner-loop problems require the same number of generations to solve. A convergence criterion is therefore imposed such that the cooperative algorithm will run until any short run of either PSO or DE fails to improve the fitness function by more than $1.0e^{-6}$. Simple problems therefore converge quickly after two or three runs of PSO or DE, and more complex problems may take ten or more runs. Second, not all inner-loop sequences are worth evaluation. For example, for an Earth to Jupiter mission the sequence EVVNJ (Earth-Venus-Venus-Neptune-Jupiter) is unlikely to be worth evaluating because the spacecraft is likely to waste more propellant and time getting to Neptune than would be gained from a gravity-assist there. The cooperative algorithm will therefore automatically quit after one run of the PSO if that first run does not find a fitness value below a certain threshold. The cooperative algorithm is described in Algorithm 4.3.

While COOP proved to be a very effective inner-loop optimizer for MGA missions, it was not as effective at optimizing MGA-DSM missions. Instead, DE or PSO alone were found to outperform COOP for MGA-DSM problems. It is not apparent why this is the case, but Vinkó and Izzo [15] made the same observation. Therefore COOP is not used to optimize MGA-DSM missions. DE and PSO work equally well. In this work, DE was used as the optimizer for MGA-DSM problems.

Algorithm 4.3 Cooperative Algorithm of DE and PSO (COOP)

generate P, a set of random position vectors $\mathbf{u}_i$ in the decision space
$count = 0$
while not hit stop criterion **do**
  if *count* is an even number **then**
    generate $\mathbf{P}_\nu$ a set of random velocity vectors ν_i in the decision space
    run PSO for $10n$ generations using population $\mathbf{P}$ and velocities $\mathbf{P}_\nu$
  else
    run DE for $10n$ generations using population $\mathbf{P}$
  end if
  increment *count*
end while
return $\mathbf{u}_{global-best}$

4.2.5 Automated Choice of Inner-Loop Bounds

Upper and lower bounds must be chosen for each decision variable. In most trajectory design problems, bounds are chosen using an analyst's knowledge or intuition about the problem. However, in the case of a hybrid optimal control problem, each inner-loop trajectory optimization problem is generated dynamically by the outer-loop. There is no opportunity for a human analyst to set reasonable bounds for each inner-loop problem. There are some parameters, such as launch date or stay time between journeys in a sample return mission, that may be chosen *a priori* by the user. However, in a HOCP, all of the other decision parameters cannot be bounded *a priori*. The flight times for each phase, flyby periapse altitudes, b-plane insertion angles, and burn indices, must be relevant to the current candidate flyby sequence, which is generated in real-time by the outer-loop solver. Different bounds are appropriate for different flyby sequences. Since there is no opportunity for the user to define these bounds when they are needed, it was necessary to define a set of simple laws that can be used to generate these bounds for any choice of flyby sequence. Tables 4.1 and 4.2 contain the laws used to choose bounds for MGA and MGA-DSM, respectively. Note particularly that the flight time bounds are functions of τ_i, the orbital periods of the bodies which define the endpoints of each phase.

Table 4.1: Automated Choosing of Inner-Loop Bounds for MGA Problems

Parameter	Lower Bound	Upper Bound
Launch date	user-defined	user-defined
Stay time between journeys	user-defined	user-defined
For each phase:		
Flight time	$\tau/2$ (repeated flyby of same planet)	5τ (repeated flyby of same planet)
	$0.1min\left(\tau_1, \tau_2\right)$	$1.5max\left(\tau_1, \tau_2\right)$ (outermost body has $a < 2AU$)
		$max\left(\tau_1, \tau_2\right)$ (outermost body has $a \geq 2AU$)
	maximum of 600 days	minimum of 1000 days

Table 4.2: Automated Choosing of Inner-Loop Bounds for MGA-DSM Problems

Parameter	Lower Bound	Upper Bound
Launch date	user-defined	user-defined
Stay time between journeys	user-defined	user-defined
RLA	0.0	2π
DLA	user-defined	user-defined
$v_{\infty-launch}$	0.0	user-defined
For each phase:		
Flight time	$\tau/2$ (repeated flyby of same planet)	5τ (repeated flyby of same planet)
	$0.1min\left(\tau_1, \tau_2\right)$	$1.5max\left(\tau_1, \tau_2\right)$ (outermost body has $a < 2AU$)
		$max\left(\tau_1, \tau_2\right)$ (outermost body has $a \geq 2AU$)
	maximum of 600 days	minimum of 1000 days
Burn index η	0.1	0.9
B-plane insertion angle γ	$-\pi$	π
Flyby periapse distance ratio R_p	1.05	10 (for rocky planets)
		300 (for gas giants)

Chapter 5

Example Optimized Impulsive-Thrust Missions

5.1 Galileo-Like Mission with no Deep Space Maneuvers

In this first of three examples, the HOCP automaton is applied to the problem of traveling from Earth to Jupiter, with insertion into a highly elliptical orbit about Jupiter at the end of the journey, modeled as an MGA problem. This type of orbit insertion was used for NASA's 1989 Galileo mission. Several flybys of Jupiter's moons were then used to reach the desired science orbit [8]. The moon flybys are not considered in this test case, but the parameters of the initial (arrival) orbit about Jupiter, as well as the launch window and upper bound on the total flight time, are the same as in the real Galileo mission. Table 5.1 contains the problem assumptions for the Earth to Jupiter mission.

Table 5.1: Problem Assumptions for the Galileo MGA Mission

Option	Value
Arrival type	insertion into orbit about Jupiter
Semi-major axis of capture orbit about Jupiter	176338 km
Eccentricity of capture orbit about Jupiter	0.998
Launch window open date	1/1/1986
Launch window close date	1/1/1994
Flight time upper bound	10 years
Maximum number of flybys	8

The EMTG was run ten times on a 2.4 GHz Intel Core 2 Duo. Average execution time was 3.5 hours per run. In all ten test runs, the EMTG converged to the sequence Earth-Venus-Venus-Earth-Earth-Jupiter (EVVEEJ). This sequence was found to have an optimal cost of 3.68 km/s. The actual Galileo mission used the EVEEJ sequence that in the MGA formulation used here has a best cost of 7.92 km/s. One cannot find a good EVEEJ solution using the MGA model because the solution used by the actual Galileo mission included a resonant flyby of Earth. Resonant flybys, i.e. scenarios where the same planet is encountered at the same point in space one or more revolutions apart, are not possible in the MGA code due to a singularity in the Lambert solver. Lambert's method finds a transfer arc between two position vectors, restricted to the plane that contains both vectors. If the two position vectors are co-linear, that occurs when the spacecraft encounters a planet twice at the same point in space, then Lambert's method cannot define the plane and therefore cannot find a transfer arc. This is an inherent limitation of the MGA model. The solution found here is therefore not representative of the actual Galileo mission, but rather represents a "Galileo-like" mission that accomplishes the same objective. Table 5.2 contains the top 20 (of 668) candidate trajectories evaluated by the outer-loop GA in a single run

of the automaton. Table 5.3 describes the optimal solution. The dates for the launch, each flyby event, and the arrival listed in Table 5.3 are the decision variables chosen by the inner-loop optimizer. The flyby altitudes and Δvs are derived from the event dates as described in Section 4.1.1. Both the flyby sequence and the trajectory are chosen entirely autonomously, with the user specifying only the parameters listed in Table 5.1. Figure 5.1 is a plot of the optimal trajectory.

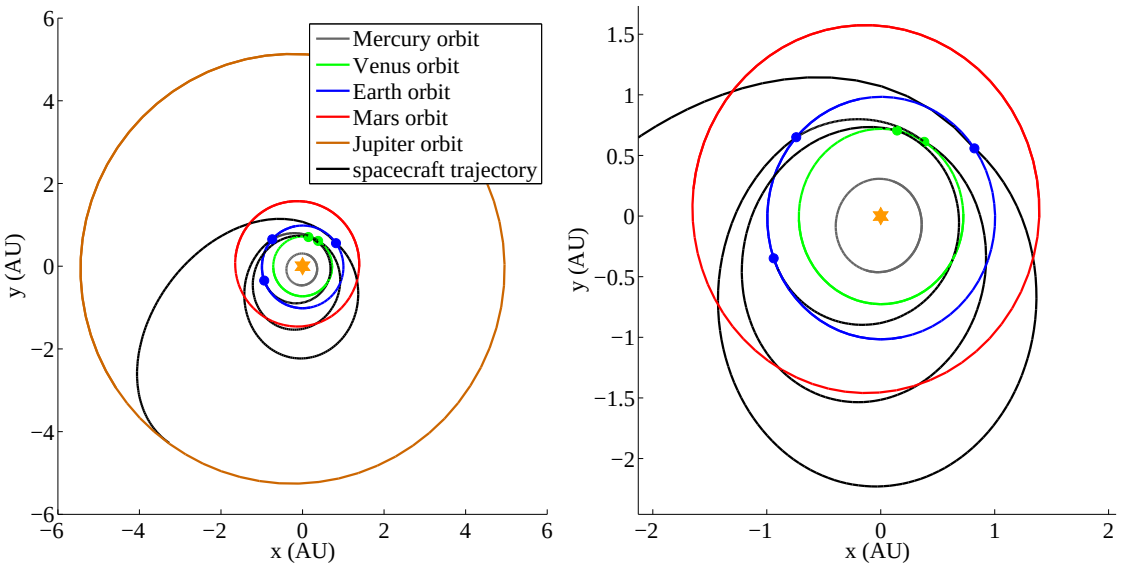
Note that with up to 8 flybys allowed and 8 possible flyby targets and accounting for the null codes, there are $\sum_{i=1}^{8} 8^i$, or 19173961 possible flyby sequences. The outer-loop algorithm removes many "bad" sequences by recognizing characteristics that have a high cost, such as doing a flyby of Uranus on the way from Earth to Jupiter. As a result, the automaton evolves towards the globally optimal flyby sequence while evaluating only a very small fraction of the possible candidates.

Table 5.2: Top 20 Candidate Sequences for the Galileo MGA Mission

Sequence	Cost (km/s)
EVVEEJ	3.68
EVVEJ	4.72
EVVEEEJ	5.24
EVVVEJ	5.52
EVEJ	5.83
EVVEVJ	5.91
EEMEJ	5.96
EEVEEJ	7.03
EEMEEJ	7.38
EEEEJ	7.61
EEEJ	7.61
EEEEEJ	7.67
EEVVEEJ	7.77
EVVEMEJ	7.8
EVVVJ	7.91
EVEEJ	7.92
EEEVEJ	8.07
EVJ	8.14
EVVVVJ	8.26
EEJ	8.3

Table 5.3: Itinerary for the Galileo MGA Mission

Encounter Date	Planet	ΔV (km/s)	Flyby Altitude (km)
4/9/1988	Earth	3.04	N/A
10/1/1988	Venus	0.07	3963
12/12/1989	Venus	0	4164
2/7/1990	Earth	0	7507
10/28/1991	Earth	0.13	1078
10/13/1994	Jupiter	0.44	N/A



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 5.1: Trajectory for the Optimal MGA Galileo Mission

5.2 Cassini-Like Mission with no Deep Space Maneuvers

The second example is an MGA Earth to Saturn mission with insertion into a highly elliptical orbit about Saturn at the end of the journey. This is similar to the 1997 NASA/ESA Cassini mission [10]. As in the case of Galileo, Cassini was inserted into a highly elliptical orbit and then moon flybys were used to reach the final science orbit. The moon flybys are not modeled in this example. Table 5.4 contains the problem

assumptions for the Cassini MGA mission.

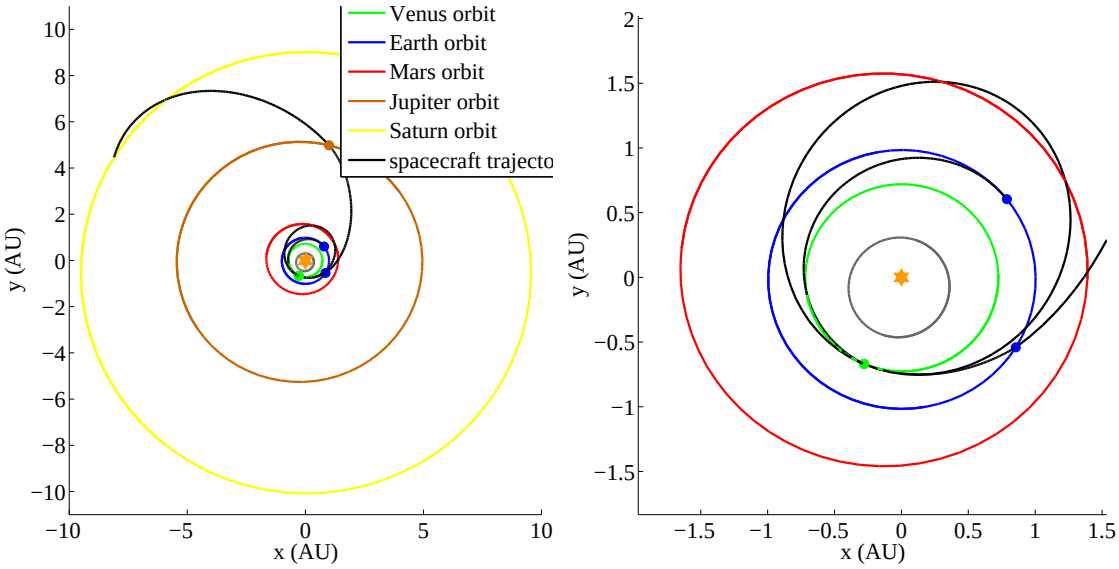
Table 5.4: Problem Assumptions for the Cassini MGA Mission

Option	Value
Arrival type	insertion into orbit about Saturn
Semi-major axis of capture orbit about Saturn	5447500 km
Eccentricity of capture orbit about Saturn	0.998
Launch window open date	4/7/1997
Launch window close date	1/1/2000
Flight time upper bound	10 years
Maximum number of flybys	8

The EMTG was run ten times on a 2.4 GHz Intel Core 2 Duo. Average execution time was 3.5 hours per run. In three of ten test runs, the automaton converged to the sequence Earth-Venus-Venus-Earth-Jupiter-Saturn (EVVEJS) - the same sequence used by the actual Cassini mission. This sequence was found to have an optimal cost of 5.98 km/s. This is higher than the best known cost of 4.93 km/s for the "Cassini1" benchmark problem described in the literature [15], but the benchmark problem does not include an upper bound on flight time and in fact the published optimal solution has a flight time of 17 years. Since this is impractical due to the limited lifetime of a spacecraft's power supply, an upper bound of 10 years was used in this study. Table 5.6 contains parameters of the optimal solution. Table 5.5 contains the top 20 (of 767) candidate trajectories evaluated by a single run of the automaton. Figure 5.2 is a plot of the optimal trajectory. The left panel of Figure 3 shows the entire trajectory, and the right panel shows a magnified view of the inner planet flybys. Note in Table 5.6 that a large burn is required at the second Venus flyby, two years into the mission, which is not desirable for a spacecraft's on-board propulsion system. If the spacecraft were not constrained to perform burns only at flyby periapse and were allowed to perform the burn at a different point along the trajectory, the cost would be lower. The actual trajectory of the Cassini mission used such mid-course burns.

Table 5.5: Top 20 Candidate Sequences for the Cassini MGA Mission

Sequence	Cost (km/s)
EVVEJS	5.98
EMEJS	7.09
EVJS	7.35
EVVES	7.39
EVVEES	7.57
EMES	7.8
EEMES	7.85
EVVJS	7.87
EMEES	8.48
EVMEEJS	8.66
EEJS	8.81
EVES	9.11
EVEJS	9.2
EEES	9.37
EJS	9.48
EES	9.68
EMEVES	9.73
EVVVJS	9.83
EVVS	10.01
EEEJS	10.13



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 5.2: Trajectory for the Optimal MGA Cassini Mission

Table 5.6: Itinerary for the Cassini MGA Mission

Encounter Date	Planet	ΔV (km/s)	Flyby Altitude (km)
5/3/1997	Earth	2.77	N/A
4/7/1998	Venus	0.97	609
7/1/1999	Venus	1.77	4012
8/21/1999	Earth	0	2822
4/18/2001	Jupiter	0	4394296
10/31/2007	Saturn	0.47	N/A

5.3 Cassini with Deep Space Maneuvers

In this third example, the Cassini mission is optimized again using the MGA-DSM structure to see if the cost can be improved. In the MGA model, the constraint that impulsive maneuvers may occur only at the periapse of flybys is relaxed, and a deep space maneuver is permitted at any point between each pair of flybys. It was expected that the addition of a deep space maneuver would allow a smaller total Δv because flyby periapse is not necessarily the optimal location for a maneuver. The same problem assumptions are used as in the MGA case, except that a maximum flight time of seven years is used instead of ten to better correspond to the flight time of the real Cassini mission. Also, to most accurately model a real interplanetary mission, ΔV_{lv}, the cost of the initial impulse from the launch vehicle, was removed from the cost function. The cost function therefore represents only the Δv provided by the spacecraft's thrusters. This is done because in a real deep space mission, there is a known relationship between ΔV_{lv} and spacecraft mass for every launch vehicle. Unfortunately the Titan IVb used to launch the Cassini mission is no longer in production and so the function defining its performance could not be found. Instead, the ΔV_{lv} value of 4.25081 km/s used in the actual Cassini mission was used as an upper bound on ΔV_{lv} in this example. Since ΔV_{lv} is bounded, it need not be represented in the cost function.

The EMTG was run in parallel on a four-core 3.3 GHz Intel Core i7 and converged

after 45.6 hours. The best sequence found was EVVEJS, the same sequence found by the MGA solver in the previous example and used in the actual Cassini mission. The best cost found was 1.01 km/s. Table 5.7 describes the optimal solution with burns less than 1 m/s omitted. Note that the optimizer chose to use the entire 4.25081 km/s of allowed ΔV_{lv}. This is unsurprising, and means that the optimal solution uses all of the launch vehicle capability available.

Table 5.7: Itinerary for the Cassini MGA-DSM Mission

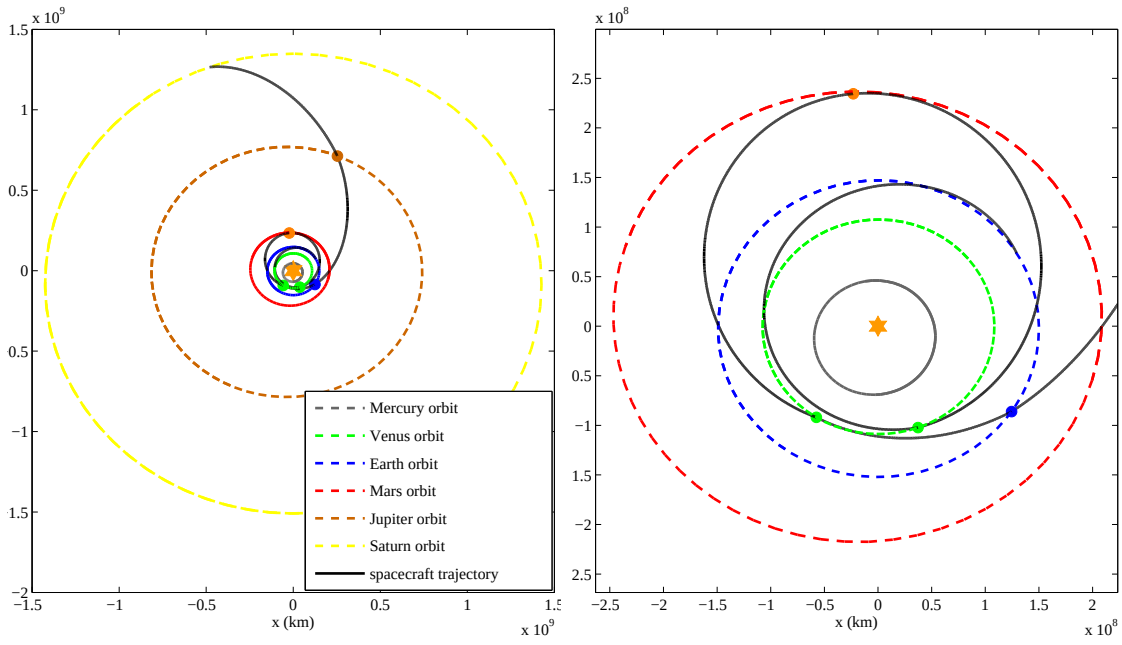
Date	Event	Location	ΔV (km/s)	Flyby altitude (km)	γ (degrees)
10/22/1997	launch	Earth	4.25081	-	-
5/3/1998	flyby	Venus	-	3952	-97.7
12/4/1998	burn	deep-space	0.389606	-	-
6/23/1999	flyby	Venus	-	728	-112.8
8/17/1999	flyby	Earth	-	924	-88.9
1/15/2001	flyby	Jupiter	-	9.53E+06	-88.8
10/22/2004	insertion	Saturn	0.619974	-	-

The optimal solution is compared to the actual Cassini mission in Table 5.8. Note that the two missions correspond very closely. There are discrepancies of a few days in the dates of the Jupiter flyby and the Saturn arrival, but they are small compared to the seven year total flight time. The earlier Jupiter flyby date in the actual Cassini mission sets up the earlier Saturn arrival date. The discrepancy is likely due to a constraint in the actual Cassini mission to perform flybys of Titan and Phoebe upon arrival at Saturn. These satellite flybys are not modeled here, so the optimizer does not allow time for Titan and Phoebe to move into the necessary positions. Also, the two-body model used in this work is only an approximation of the real N-body solar system and some deviations from a full-fidelity trajectory are expected. In just two days of run time, and with no *a priori* information about the optimal sequence of flybys, the automaton was able to reproduce the trajectory used by Cassini. This is a significant achievement compared to the large amount of work that was necessary to design the original Cassini trajectory [10]. The total cost of the mission, including the

launch vehicle impulse, is 5.26 km/s. This value is slightly better than the total ΔV of 5.33 km/s and spacecraft ΔV of 1.079 km/s used by the actual Cassini mission. Figure 5.3 shows the trajectory.

Table 5.8: Comparison of the Optimal MGA-DSM Cassini to the Actual Cassini Mission

Event	Optimal MGA-DSM Solution	Actual Cassini Mission
Launch	10/22/1997	10/15/1997
Venus flyby 1	5/3/1998	4/26/1998
Venus flyby 2	6/23/1999	6/24/1999
Earth flyby	8/17/1999	8/18/1999
Jupiter flyby	1/15/2001	12/30/2000
Saturn orbit insertion	10/22/2004	7/1/2004



(a) Full Trajectory (b) Inner-Planet Flybys Only

Figure 5.3: Trajectory for the Optimal MGA-DSM Cassini Mission

Chapter 6

Inner-Loop Optimization (of Continuous-Thrust Trajectories)

6.1 Introduction

While most interplanetary missions make use of simple, reliable chemical propulsion, there are some types of missions that cannot be easily accomplished using chemical motors. This is because while chemical motors usually provide a high thrust, they suffer from a low specific impulsive (I_{sp}). Specific impulse is related to exhaust velocity c by,

$$c = I_{sp} g_0 \tag{6.1}$$

where g_0 is the acceleration due to gravity at sea level on Earth, 9.80665 m/s^2. Exhaust velocity is related to the ratio of final mass m_{final} to initial mass $m_{initial}$, or mass fraction, by Tsiolkovsky's rocket equation [55]:

$$\frac{m_{final}}{m_{initial}} = e^{-\Delta v/c} \tag{6.2}$$

Equation 6.2 is an exponential function of Δv and exhaust velocity c, so a lower exhaust velocity, and thus a lower specific impulse, results in a higher propellant consumption for a given Δv. Figure 6.1 shows the relationship between Δv and mass fraction for two different values of I_{sp} - 300 seconds, characteristic of a chemical thruster, and 3000 seconds, characteristic of a solar electric propulsion (SEP) system that uses a high-power electric field to accelerate charged ions at high velocity instead

of a chemical reaction. It is clear that for missions that require a high Δv, the low I_{sp} of chemical propulsion is a significant disadvantage. In extreme cases, the Δv and therefore the propellant requirements may be so high that there is little mass budget for scientific instruments on the spacecraft, so that the mission has little utility.

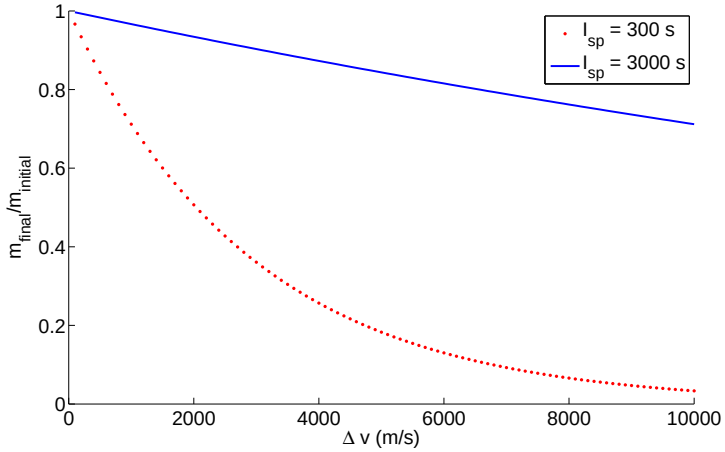


Figure 6.1: Mass Fraction as a Function of Δv and I_{sp}

Several types of missions, some of which have been flown and some of which have only been proposed, require a very high Δv and therefore cannot use chemical propulsion. For example, The Japanese Aerospace Exploration Agency (JAXA)'s Hayabusa mission to return samples from the near-Earth asteroid Itokawa used SEP because of the high total Δv required to rendezvous with and then return from the asteroid [5]. Similarly, NASA's Dawn spacecraft used SEP to rendezvous with and study the main belt asteroid Vesta and is currently on its way to Ceres [6]. Without SEP, it would not be possible to orbit both bodies. In addition, missions to the outer planets have been proposed using a combination of SEP and multiple flybys of Venus and Earth to gain enough energy to reach Jupiter, Saturn, or even Uranus and Neptune [60, 61].

Unfortunately, SEP suffers from two major disadvantages. The first disadvantage is that SEP uses solar arrays, which in turn are reliant on photons from the Sun. The farther the spacecraft is from the sun, the less power is available, and therefore

the thrust and I_{sp} of the engine degrades. This problem can be avoided by using radioisotope electric propulsion (REP), which consists of an electric motor coupled with a radiothermal generator. No REP mission has ever been flown, although REP has been proposed for missions to the outer planets [3]. The second disadvantage of SEP is unfortunately shared by REP: both systems provide a very limited amount of power, measured in hundreds or thousands of watts. This amount of power is sufficient to operate engines with specific impulses in the range of 1000 to 5000 seconds, but with very low thrust. This problem may be solved using nuclear electric propulsion (NEP), which couples a nuclear reactor with a very powerful electric thruster. The NEP system proposed for the Jupiter Icy Moons Orbiter [2] would have provided a thrust of 2.26 N at an I_{sp} of 6000 s, compared to the SEP system used by Dawn that provides a thrust of only 90 mN at an I_{sp} of 3000 s. NEP would therefore allow for much higher Δvs than any current form of propulsion. However, NEP and even REP are considered to be very expensive to build and neither has been flown.

The process of optimizing SEP, REP, and NEP missions is very different from that used to optimize chemical propulsion missions. This is because electric propulsion systems provide a thrust that is measured in fractions of a Newton, or in the most powerful case a few Newtons, whereas chemical propulsion systems provide tens, hundreds, or even thousands of Newtons of thrust. Unlike for chemical propulsion, the time spent thrusting by an SEP, REP, or NEP spacecraft is a significant fraction of the total flight time. The impulsive approximation, i.e. that the trajectory is a sequence of Keplerian arcs, introduced in Chapter 4 is therefore not applicable. Instead, the SEP, REP, and NEP missions fall into a general category of "continuous-thrust" missions. In continuous-thrust mission design, a time history of thrust magnitude and direction must be chosen that brings the spacecraft to its destination while optimizing a cost function, usually maximizing the final mass. The spacecraft no longer follows a strict Keplerian path, and therefore Kepler and Lambert models used in Chapter 4 are no

longer accurate. Continuous-thrust mission optimization therefore requires a new set of physical models and optimization methods.

There are two families of approaches to this problem: direct and indirect. In indirect trajectory optimization, the trajectory design is formulated as an optimal control problem and analytic necessary and sufficient conditions are derived in terms of the states and Lagrange multipliers, which represent the sensitivity of the solution to the constraints. The problem is then solved as a multi-point boundary value problem [20]. Some indirect optimization problems are small enough, in terms of the number of variables and constraints, that they may be solved analytically. However spacecraft trajectory optimization problems usually have a large number of variables and constraints and therefore must be solved numerically. A nonlinear programming (NLP) problem solver may be used to find the optimal value of the Lagrange multipliers. This approach requires a good initial guess. Trajectory optimization problems are very sensitive to this initial guess, and in the case of indirect optimization the guess can be very non-intuitive.

In direct optimization, the problem is parameterized by physically meaningful decision variables such as flight times, thrust magnitudes and control angles, flyby altitudes, etc. rather than by Lagrange multipliers. Many different transcription methods are used with direct solvers, including direct collocation with a polynomial approximation to the control parameters [21], direct transcription [22], perturbative methods [23], and approximation of the continuous trajectory as a series of small coast arcs punctuated by small impulses [24]. Like indirect methods, direct methods require a NLP problem solver and a good initial guess of the decision variables.

In a mission design automaton such as the one presented in this work, it is necessary to use a method that simultaneously provides as high a level of fidelity as possible but can also be applied rapidly and reliably. Speed is important because, in an automated method, hundreds or even thousands of candidate flyby sequences might be

evaluated and if each trajectory takes days to optimize, then the automaton will be too slow to be useful. Reliability is important because if the inner-loop trajectory optimizer of the automaton fails to return a valid cost metric for a candidate flyby sequence, then the automaton will halt. The inner-loop solver must either provide a feasible solution or a measurement of infeasibility for every candidate.

In this work, direct methods are used instead of indirect methods for two reasons. First, indirect methods are difficult to formulate. Analytical necessary and sufficient conditions must be derived for each problem, and then must be re-derived if the structure of the problem changes. The HOCP automaton is intended to be a flexible tool that can be used for a variety of mission types, and so the requirement to re-derive equations is undesirable. In addition, the structure of the inner-loop trajectory optimization problem is generated in real time by the outer-loop. It is therefore not possible for a human analyst to re-derive analytical necessary and sufficient conditions for every possible problem. It may be possible to construct the required equations in an automated fashion, but this would be much more challenging than using a direct method. By contrast, it is very easy to add or remove constraints from a direct method. The user, or in this case the automaton, must only specify an additional constraint to the nonlinear programming solver.

The second disadvantage of indirect methods is that finding a suitable initial guess and lower and upper bounds for the Lagrange multipliers is very difficult. The Lagrange multipliers represent sensitivities of the constraints to each of the decision variables instead of physical quantities, and therefore their values, and even their orders of magnitude, are not intuitive. It would be very challenging to develop a reliable initial guess method to find the Lagrange multipliers, and therefore very difficult to create an automated mission design tool using indirect methods.

6.2 Direct Methods for the Continuous-Thrust Trajectory Optimization Problem

In a continuous-thrust trajectory optimization problem, the vehicle moves under the influence of an inverse-square gravity field and a thrust term that represents the influence of the propulsion system. In a space-fixed Cartesian basis, the system governing equations are:

$$\ddot{x} = \frac{-\mu}{r^3}x + \frac{T_x}{m} \tag{6.3}$$

$$\ddot{y} = \frac{-\mu}{r^3}y + \frac{T_y}{m} \tag{6.4}$$

$$\ddot{z} = \frac{-\mu}{r^3}z + \frac{T_z}{m} \tag{6.5}$$

$$\dot{m} = -\frac{T}{I_{sp}g_0} \tag{6.6}$$

where μ is the gravitational parameter of the central body, in this case the Sun, r is the distance from the central body, g_0 is the acceleration due to gravity at sea level on the Earth, and T is the magnitude of the thrust vector,

$$r = \sqrt{x^2 + y^2 + z^2} \tag{6.7}$$

$$T = \sqrt{{T_x}^2 + {T_y}^2 + {T_z}^2} \tag{6.8}$$

The equations of motion must be integrated forward in time to find the correct trajectory. The optimizer must choose the continuous time history of the thrust vector $\mathbf{T}$. The thrust magnitude T is bounded in $[0, T_{\max}]$. It is very challenging to find an optimal continuous function for T, and so approximation methods are often used that define the thrust vector at a set of control points and interpolate to find the thrust vector between control points. These methods include Direct Collocation with Nonlinear Programming (DCNLP) [21] and Direct Transcription with Nonlinear

Programming (DTNLP) [22].

Other methods exist that require further simplifying assumptions about the trajectory. The most notable of these is the Sims-Flanagan transcription, which models the continuous-thrust trajectory as a set of small impulses equally spaced in time [24]. If a sufficient number of impulses is used, then the Sims-Flanagan model closely approximates a true continuous-thrust trajectory. The Sims-Flanagan transcription has been used successfully to design multiple-flyby, continuous-thrust missions in the past [2, 3, 33, 34] but was initially considered unsuitable in this work because of its impulsive approximation.

6.3 Direct Transcription with Nonlinear Programming (DTNLP)

The choice of direct method for this work was a process of trial and error. The first method applied was Direct Transcription with Nonlinear Programming, chosen because it is known to be robust for simple continuous-thrust optimization problems and because the only approximation that it requires is that the control variables are defined at discrete node points. Unlike in the Sims-Flanagan model, the continuous motion of the spacecraft is modeled.

In DTNLP, the continuous-time trajectory optimization problem is discretized into time segments $[1, 2, 3, ..., N_{segments}]$, each of length h. The time history of each state and control value is specified at the $N_{segments} + 1$ node points between segments as shown in Figure 6.2. In addition, the control variables are specified at interior points within each segment, as shown in Figure 6.3. The equations of motion are then integrated across each time step using a 4th-order, 3-step Runge-Kutta integration rule.

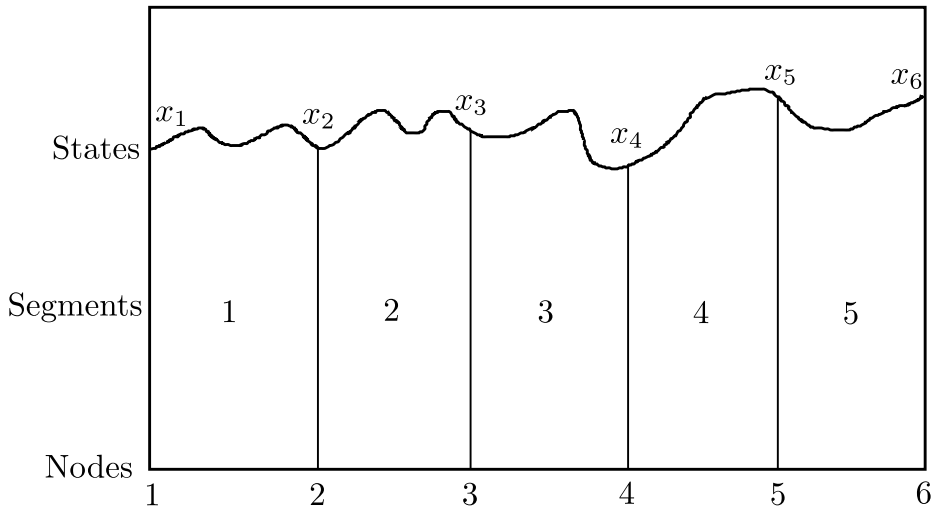


Figure 6.2: Problem Structure for DTNLP

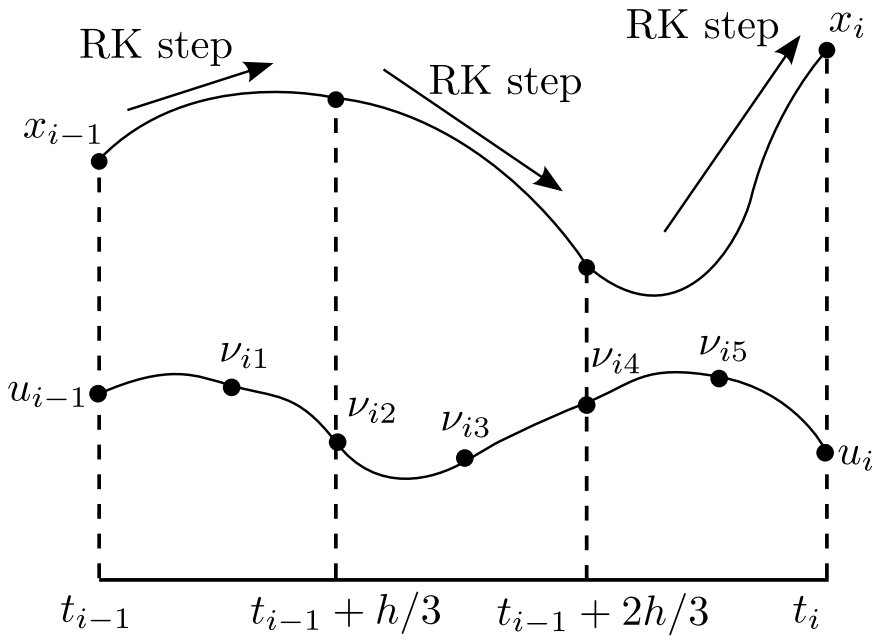


Figure 6.3: 3-Step Runge-Kutta Integration Rule for one Time Segment

In a feasible solution, the time histories of the state variables x and the control variables u must satisfy the governing equations. This can be done by explicit numerical integration across the entire problem, but this method is computationally expensive. Instead, implicit integration is used. An analytical set of constraints is found that, if satisfied, guarantee that the equations of motion are satisfied across the entire problem. To generate these constraints, the state variables are integrated

across each time step from the left hand node to the right hand node, using the interior control points $\nu_{i1}...\nu_{i5}$ as shown in Figure 6.3. The integration is done in three steps in Figure 6.3, each using the 4th order Runge-Kutta rule shown below. More or fewer steps could be used in an alternate formulation. The following equations are specific to the leftmost step, but can easily be generalized to the other two steps.

$$y_1 = x_{i-1} + \frac{h}{2} f\left(x_{i1}, u_{i1}\right) \tag{6.9}$$

$$y_2 = x_{i-1} + \frac{h}{2} f\left(y_1, \nu_{i1}\right) \tag{6.10}$$

$$y_3 = x_{i-1} + \frac{h}{2} f\left(y_2, \nu_{i1}\right) \tag{6.11}$$

$$y_4 = x_{i-1} + \frac{h}{6}\left[f\left(x_{i-1}, u_{i1}\right) + 2f\left(y_1, \nu_{i1}\right) + 2f\left(y_2, \nu_{i1}\right) + f\left(y_3, \nu_{i2}\right)\right] \tag{6.12}$$

Here $f\left(x, u\right)$ refers to the system equations of motion, x are the state variables, u are the control variables, and y_4 are the integrated values of the state at the right hand side of the integration step. For example, y_4 for the first step represents $x\left(t_{i-1} + h/3\right)$. This value is then used as the initial condition x_{i-1} for the next step, and then y_4 for the second step is used as the initial condition for the third step. The control variables u_i and ν_{ik} are specified at the five interior points, but the state variables x are not. This significantly decreases the size of the problem, increasing solution speed without sacrificing accuracy [22]. The difference between the integrated state at the right hand side of each time step x_i^* (y_4 for the third step), and the NLP parameter for the value x_i of the state at the left node of the next segment forms the nonlinear defect equation:

$$\Delta_i = x_i^* - x_i \tag{6.13}$$

The set of the defect equations for all of the states over all of the time segments forms a vector of nonlinear constraints that are zero only if the equations of motion are

satisfied. If they are satisfied, then the equations of motion are effectively integrated across the entire problem using the 4th order Runge-Kutta method. Finally, a set of terminal constraints are used to ensure that the position of the spacecraft matches that of each flyby target at the moment of flyby, and also that the position and velocity of the spacecraft at the end of the journey match that of the destination planet.

An NLP solver may be used to find a trajectory that satisfies the constraint equations and optimizes a performance metric such as minimum Δv or maximum propellant delivered to a destination. However, NLP solvers require a good initial guess of the solution. This is important for two reasons. First, if a very poor initial guess is provided, the NLP solver will have difficulty finding a solution that satisfies the constraints. Second, even if a feasible solution is found, it will tend to be locally optimal in the neighborhood of the initial guess. To find a globally optimal solution, an initial guess in the neighborhood of the global optimum must be used.

There are several different methods of finding initial guesses for continuous-thrust optimization problems, including constant thrust initial guesses [21], ballistic initial guesses [25], shape based methods [26–28], and Feasible Region Analysis (FRA) [41, 45]. All of these methods work by reducing the number of variables necessary to define the continuous-thrust trajectory so that an approximation to the optimal trajectory may be determined using intuition, a grid search, or evolutionary algorithms. Constant thrust and ballistic initial guesses are the simplest form. In a constant thrust initial guess, the spacecraft is assumed to thrust directly along or against the velocity vector. Unfortunately, while this is often a good choice for simple orbit transfers and escape trajectories, globally optimal interplanetary trajectories, especially those with flybys, are not necessarily in the neighborhood of a constant thrust guess. Ballistic initial guesses consist of using the MGA or MGA-DSM formulation as an initial guess for a continuous-thrust problem. They are not ideal because, while they may yield

feasible trajectories, sometimes the globally optimal continuous-thrust trajectory is not in the neighborhood of the globally optimal MGA or MGA-DSM solution. However, shape-based methods and FRA can be used to find more complex trajectories and are therefore considered in this work to find initial guesses for DTNLP.

6.4 Inverse-Polynomial Shape-Based Approximation to Continuous-Thrust Trajectories

The first method used to find an initial guess for the DTNLP solver was a shape-based approximation method developed first in two dimensions by Wall and Conway [27], and later improved by Wall and Novak to model three dimensional problems [62]. The shape-based approximation method assumes that the continuous-thrust trajectory takes a certain shape that can be parameterized by a small number of values and has endpoints at the desired starting and ending bodies of the continuous-thrust arc. This shape always satisfies the equations of motion and the terminal constraints for each arc.

The shape-based approximation here is defined in terms of the general framework defined by Novak and Vasile [28]. Before defining the shape itself, it is necessary to define a coordinate system in which the derivation can occur. This coordinate system is defined by the unit vector in the direction of the spacecraft velocity $\hat{\mathbf{e}}_v$, the unit vector normal velocity vector in the instantaneous orbit plane $\hat{\mathbf{e}}_n$, and the instantaneous angular velocity unit vector $\hat{\mathbf{e}}_h$. The derivation also requires the spherical coordinate unit vectors $(\hat{\mathbf{e}}_r, \hat{\mathbf{e}}_\theta, \hat{\mathbf{e}}_\phi)$. The unit vectors used in the shape-based method are defined in Figure 6.4.

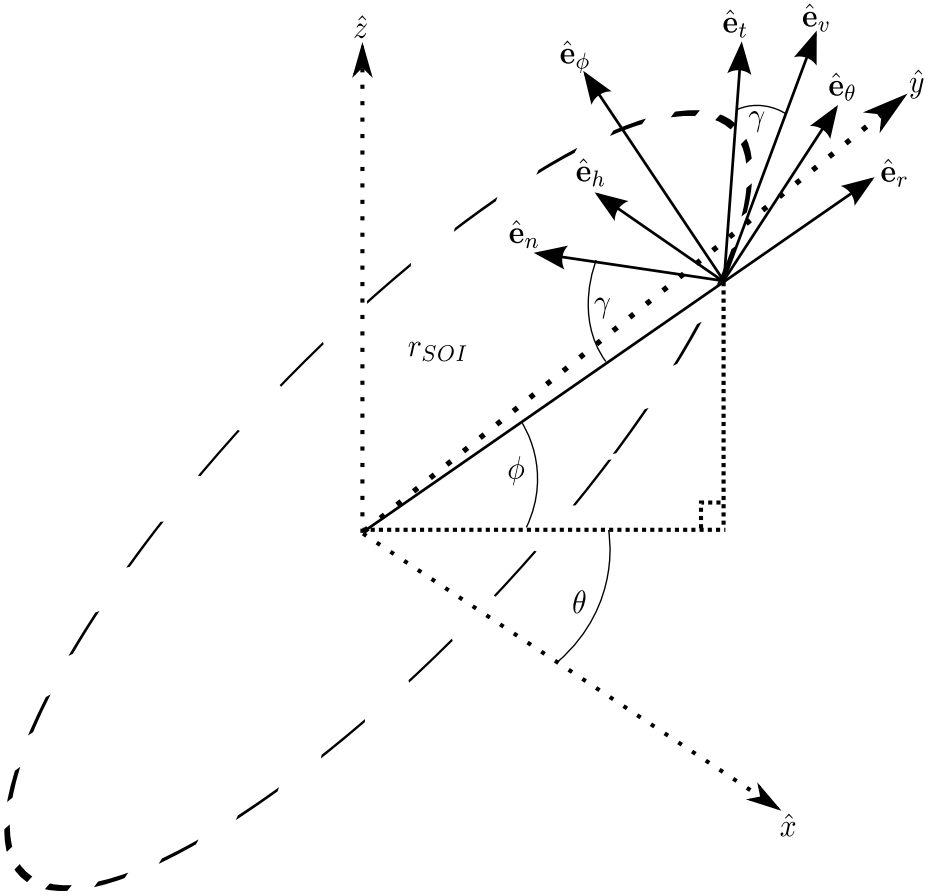


Figure 6.4: Unit Vectors Used in the Shape-Based Trajectory Approximation Method

From the position vector, the velocity vector is defined as a function of θ:

$$\mathbf{r} = r\hat{e}_r \tag{6.14}$$

$$\mathbf{v} = \frac{d\mathbf{r}}{d\theta}\dot{\theta} = \widetilde{\mathbf{v}}\dot{\theta} \tag{6.15}$$

where $\frac{d\mathbf{r}}{d\theta}$ is defined as $\widetilde{\mathbf{v}}$. The acceleration vector is then be derived and set equal to the equations of motion for the spacecraft in the inertial reference frame:

$$\mathbf{a} = \frac{d^2\mathbf{r}}{d\theta^2}\dot{\theta}^2 + \frac{d\mathbf{r}}{d\theta}\ddot{\theta} = \widetilde{\mathbf{a}}\dot{\theta}^2 + \widetilde{\mathbf{v}}\ddot{\theta} = -\frac{\mu}{r^3}\mathbf{r} + \mathbf{u} \tag{6.16}$$

Equation 6.16 may then be solved for the control vector $\mathbf{u}$:

$$\mathbf{u} = \widetilde{\mathbf{a}}\dot{\theta}^2 + \widetilde{\mathbf{v}}\ddot{\theta} + \frac{\mu}{r^3}\mathbf{r} \tag{6.17}$$

The dot product of $\mathbf{u}$ is taken with the unit vectors $(\hat{\mathbf{e}}_v, \hat{\mathbf{e}}_n, \hat{\mathbf{e}}_h)$,

$$\mathbf{u} \cdot \begin{bmatrix} \hat{\mathbf{e}}_v \\ \hat{\mathbf{e}}_n \\ \hat{\mathbf{e}}_h \end{bmatrix} = \begin{bmatrix} u_v \\ u_n \\ u_h \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{\mathbf{e}}_v + \widetilde{\mathbf{v}}\ddot{\theta} \cdot \hat{\mathbf{e}}_v + \frac{\mu}{r^3}\mathbf{r} \cdot \hat{\mathbf{e}}_v \\ \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{\mathbf{e}}_n + \widetilde{\mathbf{v}}\ddot{\theta} \cdot \hat{\mathbf{e}}_n + \frac{\mu}{r^3}\mathbf{r} \cdot \hat{\mathbf{e}}_n \\ \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{\mathbf{e}}_h + \widetilde{\mathbf{v}}\ddot{\theta} \cdot \hat{\mathbf{e}}_h + \frac{\mu}{r^3}\mathbf{r} \cdot \hat{\mathbf{e}}_h \end{bmatrix} \tag{6.18}$$

and then, using the fact that the dot product of perpendicular vectors is zero,

$$\begin{bmatrix} u_v \\ u_n \\ u_h \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{e}_v + \widetilde{\mathbf{v}}\ddot{\theta} \cdot \hat{\mathbf{e}}_v + \frac{\mu}{r^3}\mathbf{r} \cdot \hat{\mathbf{e}}_v \\ \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{\mathbf{e}}_n + \frac{\mu}{r^3}\mathbf{r} \cdot \hat{\mathbf{e}}_n \\ \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{\mathbf{e}}_h \end{bmatrix} \tag{6.19}$$

u_n may be written in terms of the unit vectors $\hat{\mathbf{e}}_r$ and $\hat{\mathbf{e}}_n$:

$$u_n = \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \hat{\mathbf{e}}_n + \frac{\mu}{r^2}\hat{\mathbf{e}}_r \cdot \hat{\mathbf{e}}_n \tag{6.20}$$

Values for the unit vectors can then be defined based on geometry:

$$\widetilde{\mathbf{h}} = \mathbf{r} \times \widetilde{\mathbf{v}} \tag{6.21}$$

$$\hat{\mathbf{e}_n} = \frac{\widetilde{\mathbf{h}} \times \widetilde{\mathbf{v}}}{\widetilde{h}\widetilde{v}} \tag{6.22}$$

$$\hat{\mathbf{e}_r} \cdot \hat{\mathbf{e}_n} = -\cos\gamma \tag{6.23}$$

$$\widetilde{h} = r\widetilde{v}\sin\left(\frac{\pi}{2} - \gamma\right) = r\widetilde{v}\cos\gamma \tag{6.24}$$

These values may be substituted into the expression for u_n:

$$u_n = -\frac{\mu}{r^2}\cos\gamma + \widetilde{\mathbf{a}}\dot{\theta}^2 \cdot \frac{\widetilde{\mathbf{h}} \times \widetilde{\mathbf{v}}}{r\widetilde{v}^2\cos\gamma} \tag{6.25}$$

Equation 6.25 may be rearranged and u_n set equal to zero to find an expression for $\dot{\theta}$ in terms of the quantity D:

$$\frac{u_n}{\cos\gamma} + \frac{\mu}{r^2} = \frac{\widetilde{\mathbf{a}} \cdot \left(\widetilde{\mathbf{h}} \times \widetilde{\mathbf{v}}\right)}{r\widetilde{v}^2\cos\gamma}\dot{\theta}^2 \tag{6.26}$$

$$r\widetilde{v}^2\cos\gamma = \frac{\widetilde{h}^2}{r} \tag{6.27}$$

$$\frac{u_n}{\cos\gamma} + \frac{\mu}{r^2} = \left[\frac{\widetilde{h}^2}{r}\widetilde{\mathbf{a}} \cdot \left(\widetilde{\mathbf{h}} \times \widetilde{\mathbf{v}}\right)\right]\dot{\theta}^2 \tag{6.28}$$

$$D = \frac{\widetilde{h}^2}{r}\widetilde{\mathbf{a}} \cdot \left(\widetilde{\mathbf{h}} \times \widetilde{\mathbf{v}}\right) \tag{6.29}$$

$$\dot{\theta} = \sqrt{\frac{\mu}{Dr^2}} \tag{6.30}$$

When u_n is set equal to zero, thrust may only be applied along or against the velocity vector, or along or against the angular momentum vector. Thrusting along or against the velocity vector allows the spacecraft to change its in-plane kinetic energy at the maximum possible rate. This strategy is not optimal but can be used to produce

good 2D trajectories [63]. By thrusting along or against the angular momentum vector, the spacecraft may effectively change the inclination of its orbit. Although both of these strategies are sub-optimal, they are a satisfactory approximation for producing initial guess trajectories between two bodies [62].

For a rendezvous or intercept problem, the time of flight must be equal to the time used to set the terminal boundary conditions. Therefore, Equation 6.30 must be integrated with respect to θ:

$$\int_0^{t_f} dt = t_f = \int_0^{\theta_f} \sqrt{\frac{Dr^2}{\mu}} d\theta \tag{6.31}$$

This integration must be performed numerically, in this case with a simple trapezoid-rule scheme.

Finally, the user must define a coordinate frame in which the control variables will be determined. In the method developed by Wall *et al.* [27, 44, 62, 63], the shape is defined in spherical coordinates (r, θ, ϕ), where r is the distance from the central body, θ is the angle defining motion in the ecliptic plane, and ϕ is the angle defining motion out of the plane. The shape of the trajectory in the (r, θ) plane is defined by,

$$r(\theta) = \frac{1}{a + b\cos(\theta + c) + d\theta^3 + e\theta^4 + f\theta^5 + g\theta^6} \tag{6.32}$$

and the out-of-plane motion is defined by,

$$\phi(\theta) = (b_0 + b_1\theta)\cos(\theta) + (b_2 + b_3\theta)\sin(\theta) \tag{6.33}$$

The endpoints of the trajectory are two fixed points in space, in this case two planets. Given known starting and ending epochs, the positions of the planets are known. Therefore the path of the spacecraft may be fully defined using the boundary conditions and Equations 6.32 and 6.33. One can then write:

$$\mathbf{r} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \tag{6.34}$$

$$\widetilde{\mathbf{v}} = \begin{bmatrix} r' \\ r\cos\phi \\ r\phi' \end{bmatrix} \tag{6.35}$$

$$\widetilde{\mathbf{a}} = \begin{bmatrix} r'' - r\left[(\phi')^2 + \cos^2\phi\right] \\ 2r'\cos\phi - 2r\phi'\sin\phi \\ 2r'\phi' + r\left(\phi'' + \sin\phi\cos\phi\right) \end{bmatrix} \tag{6.36}$$

where the prime operator signifies a derivative with respect to θ. The quantity D may then be written,

$$D = -r'' + r\left((\phi') + \cos^2\phi\right) + \frac{2\left(r'\right)^2}{r} + r'\phi'\left(\frac{\phi'' - \sin\phi\cos\phi}{\left(\phi'\right)^2 + \cos^2\phi}\right) \tag{6.37}$$

The eleven unknown coefficients in Equations 6.32 and 6.33 correspond to seven scalar constraints for the in-plane trajectory and four additional scalar constraints for the out-of-plane motion. The trajectory must begin at the first body in the phase and end at the second. Since the orbits of the bodies are known, their positions may be defined by a single angle, i.e. true anomaly, and therefore only one coefficient is required to satisfy each position constraint. Also, the trajectory must satisfy a velocity constraint at each endpoint. In a 2D trajectory, the velocity may be represented as a magnitude and a flight path angle, and so the 2D velocity constraint at each end-point is satisfied by two coefficients. Similarly, the out-of-plane trajectory must satisfy a position angle constraint and a velocity angle constraint at both end-points, requiring an additional four coefficients. Wall, Pols, and Lanktree [63] showed that these ten

terminal constraints can be used to solve algebraically to solve for ten of the eleven unknown coefficients in Equations 6.32 and 6.33. The remaining coefficient, d, is used to ensure that the trajectory matches a time of flight constraint. A root-finding algorithm is used to vary d until the flight time determined from Equation 6.31 matches that used to determine the boundary conditions. The root-finding algorithm does not always succeed, and a failure of the root-finder is considered an infeasible solution because no shape-based trajectory can be found that links the boundary points and obeys the equations of motion in the allowed flight time.

Wall *et al.* have shown that the inverse-polynomial shape-based approximation is suitable for finding initial guesses of low-thrust trajectories in two and three dimensions for both intercept (match the position of a target) and rendezvous (match both the position and velocity of a target) trajectories [27, 44, 62, 63]. The shape-based approximation has the advantage of requiring only a small number of decision variables and no explicit nonlinear constraints. Accordingly, it is very fast to compute and is a natural choice for a broad search of possible trajectories using an evolutionary algorithm (EA) or grid search.

Unfortunately, the shape-based method requires well-defined boundary conditions, including initial and final velocities for each phase. In a multiple-flyby interplanetary mission, there are many phases and the initial velocity of each successive phase is dependent on the final velocity of the phase before it as well as the geometry of the flyby maneuver that occurs between the two phases. When the shape-based trajectory approximation method was implemented as an inner-loop model in the EMTG and applied to a simple Earth-Earth-Jupiter flyby sequence, no feasible trajectories were found. This is because the optimizer was unable to choose a final velocity vector for the first phase (Earth-Earth) that set up an acceptable trajectory in the second phase (Earth-Jupiter). After this experiment, it was determined that while it may eventually be possible to use the shape-based approximation in a multiple-flyby mis-

sion design problem, it would be useful to search for other methods that might be more robust.

6.5 Feasible Region Analysis (FRA)

The second method considered for finding initial guesses was feasible region analysis, or FRA. FRA was developed by Chilan and Conway as an inner-loop solver for a hybrid optimal control automaton that could find optimal spacecraft trajectories using both impulsive and continuous thrust [41, 45]. An outer-loop binary genetic algorithm (GA) was used to choose the sequence of impulses and continuous-thrust arcs. Coast arcs were assumed to take place between each pair of propulsive events. The DTNLP method described in Section 6.3 was used to solve the inner-loop trajectory optimization problem. Because they built an HOCP automaton, Chilan and Conway faced the same challenge as in this work: the inner-loop trajectory optimization must always return a cost value, otherwise the outer-loop solver halts. If a feasible solution is found, the inner-loop returns an objective function value, such as the amount of mass delivered to the destination. If no feasible solution is found, the inner-loop may return a metric of infeasibility that will cause the outer-loop to discard the candidate solution. In this section, only the continuous-thrust capability of FRA is discussed.

FRA is a method to assure that the inner-loop returns a solution in the neighborhood of the globally optimal solution. Given two end points of the trajectory, FRA returns either a near-optimal trajectory if it is possible to reach the destination using the permitted thrust, or alternatively a measurement of how close the spacecraft is able to get to the destination if the transfer cannot be completed. This is done by using a combination of a real-valued GA and the DTNLP method.

Chilan first defines a *feasible region* - the set of all points reachable from a given initial point given a fixed flight time and bounded thrust magnitude. The *feasible*

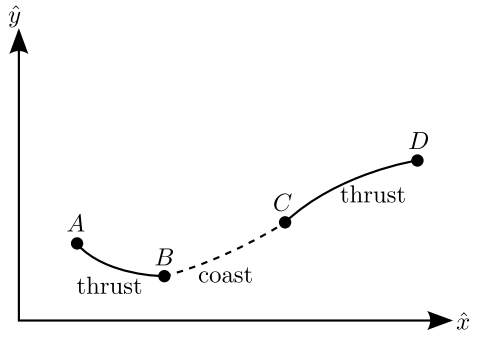


Figure 6.5: Definition of the Waypoints for Feasible Region Analysis

region for a given thrust arc is thus defined as the space of the final position and velocity vectors that are accessible. The GA is then used to choose desired boundary points for each thrust arc. An NLP solver can then be used to find the point in the feasible region for each thrust arc that is as close as possible to the desired waypoint. For example, suppose that FRA is used to find the optimal trajectory between two planets given a thrust-coast-thrust mission structure as shown in Figure 6.5. Point **A** represents the initial point for the trajectory, i.e. the first planet. Point **B** represents the end of the first thrust arc and thus the beginning of the coast arc. Point **C** represents the end of the coast arc and the beginning of the second thrust arc, and point **D** represents the destination, i.e. the second planet. Once the initial and final time are chosen, points **A** and **D** are known because the positions and velocities of the planets can be determined via an ephemeris. Point **C** can be found by integrating the equations of motion from point **B** for time $t_{\mathbf{BC}}$. The only completely undetermined waypoint is thus B. Neglecting the control variables, which are chosen later via the DTNLP method, the minimal decision vector necessary to define the trajectory is:

$$\mathbf{X} = \begin{bmatrix} t_{\mathbf{AB}} & \mathbf{r_B} & \mathbf{v_B} & t_{\mathbf{BC}} & t_{\mathbf{CD}} \end{bmatrix} \tag{6.38}$$

Here $\mathbf{r_B}$ and $\mathbf{v_B}$ are the position and velocity vectors defining point **B**, and the t_i are the flight times between waypoints.

A real-valued GA is used to choose candidate values of $\mathbf{X}$. For each candidate $\mathbf{X}$, the DTNLP method is used to find the trajectory successively for each arc. In the thrust-coast-thrust example described here, the first step is to find the point in the feasible region, **B'** that is as close as possible to the chosen waypoint **B** as shown in the cartoon of Figure 6.6. This is done by finding the time-history of control variables that minimizes the distance between **B'** and **B**. In this case, the control variables are angles defining the thrust direction. Thrust magnitude is fixed to be the maximum available to the spacecraft. The DTNLP method is thus used to solve the problem:

$$\begin{aligned} &\text{Minimize } d_{\mathbf{BB}'}(\mathbf{X}) = \|\mathbf{B} - \mathbf{B}'\| \\ &\text{Subject to:} \\ &\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{t}) \\ &\mathbf{x}(\mathbf{t_A}) = \mathbf{A} \\ &\mathbf{x}(\mathbf{t_B}) = \mathbf{B}' \end{aligned} \tag{6.39}$$

Here $\mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{t})$ is a vector function of the equations of motion, and $\mathbf{x}$ and $\mathbf{u}$ are the state and control vectors chosen by the NLP solver to solve system 6.39. This NLP problem always has a solution. A cost function value of zero means that the waypoint B is within the feasible region.

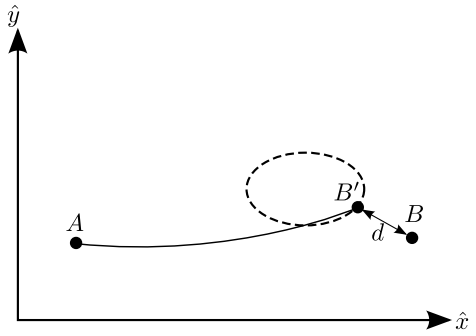


Figure 6.6: Choosing the Endpoint of a Thrust Arc in Feasible Region Analysis

Next, the equations of motion are integrated forward in time from point B' to point C using no control and flight time $t_{\mathbf{BC}}$. Finally, C becomes the initial point for

another thrust arc optimization problem that minimizes the distance $d_{\mathbf{DD'}}$ between an achievable **D'** in the feasible region and the terminal point **D** as shown in Figure 6.7. The real-valued GA is then used to find the decision vector X^* for which points **B** and **D** reside in the feasible regions of the first and second thrust arc, respectively and that minimizes the amount of time spent thrusting. I.e., the GA minimizes the metric:

$$J = \begin{cases} d_{\mathbf{DD'}} & if \quad d_{\mathbf{DD'}} > 0 \\ t_{\mathbf{AB}} + t_{\mathbf{CD}} & if \quad d_{\mathbf{DD'}} = 0 \end{cases} \tag{6.40}$$

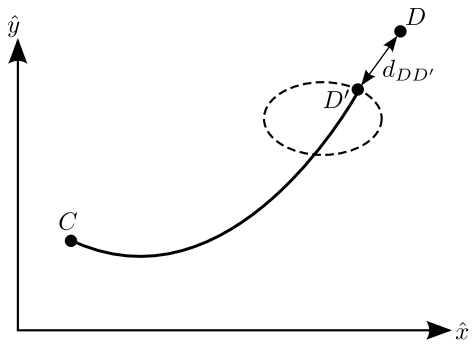


Figure 6.7: Finding the Point in the Feasible Region Closest to the Destination Planet

Feasible region analysis was used successfully by Chilan and Conway to optimize transfers from Earth to Mars and also a rendezvous problem between two spacecraft that begin at opposite sides of a circular orbit [41, 45]. Since FRA always finds a feasible trajectory, it is ideal for use in an HOCP automaton. FRA was therefore considered as an inner-loop solver for the multiple-flyby problems addressed in this work.

A test problem was constructed in which a spacecraft travels from the Earth to Jupiter using continuous-thrust propulsion and a single flyby of the Earth. The flyby was modeled using the same method as in the MGA-DSM model. Since there are two

phases in the EEJ problem, the FRA objective function is redefined as:

$$J = \begin{cases} d_{\mathbf{DD'},\mathbf{1}} + d_{\mathbf{DD'},\mathbf{2}} & if \quad d_{\mathbf{DD'},\mathbf{1}} + d_{\mathbf{DD'},\mathbf{2}} > 0 \\ t_{\mathbf{AB},\mathbf{1}} + t_{\mathbf{CD},\mathbf{1}} + t_{\mathbf{AB},\mathbf{2}} + t_{\mathbf{CD},\mathbf{2}} & if \quad d_{\mathbf{DD'},\mathbf{1}} + d_{\mathbf{DD'},\mathbf{2}} = 0 \end{cases} \tag{6.41}$$

The position of the Earth at the time of flyby is found using an ephemeris, but the velocity of the spacecraft at that time is left free. The optimizer must instead choose the altitude ratio R_p and b-plane insertion angle γ that, combined with whatever velocity that the spacecraft has upon arrival at the flyby planet, determine the outgoing velocity vector from the flyby.

Unfortunately, the GA was unable to find a solution in which $d_{\mathbf{DD'},\mathbf{1}} + d_{\mathbf{DD'},\mathbf{2}} = 0$, meaning that a feasible trajectory was not found. This is likely because in a given candidate inner-loop solution, the distance $d_{\mathbf{DD'},\mathbf{1}}$ between the end of the second thrust arc and the flyby planet, is not necessarily zero. It is only in the final converged solution that $d_{\mathbf{DD'},\mathbf{1}}$ is assured to be zero. Unfortunately, this means that the velocity vector relative to the flyby planet at the end of the first phase, $\mathbf{v}_1$, is not a meaningful concept because the spacecraft is not actually at the planet. Therefore the flyby equations are only valid for the final converged solution and not for the intermediate solutions. When the spacecraft is not in the same location as the planet, then its velocity relative to the planet is not a useful concept and the flyby cannot be computed. Figure 6.8 illustrates an example where the spacecraft does not reach the flyby planet and therefore the flyby equations yield non-physical results, in this case a false retrograde trajectory after the flyby. As a result, FRA was discarded as a potential method for generating an initial guess for the inner-loop problem solver using NLP.

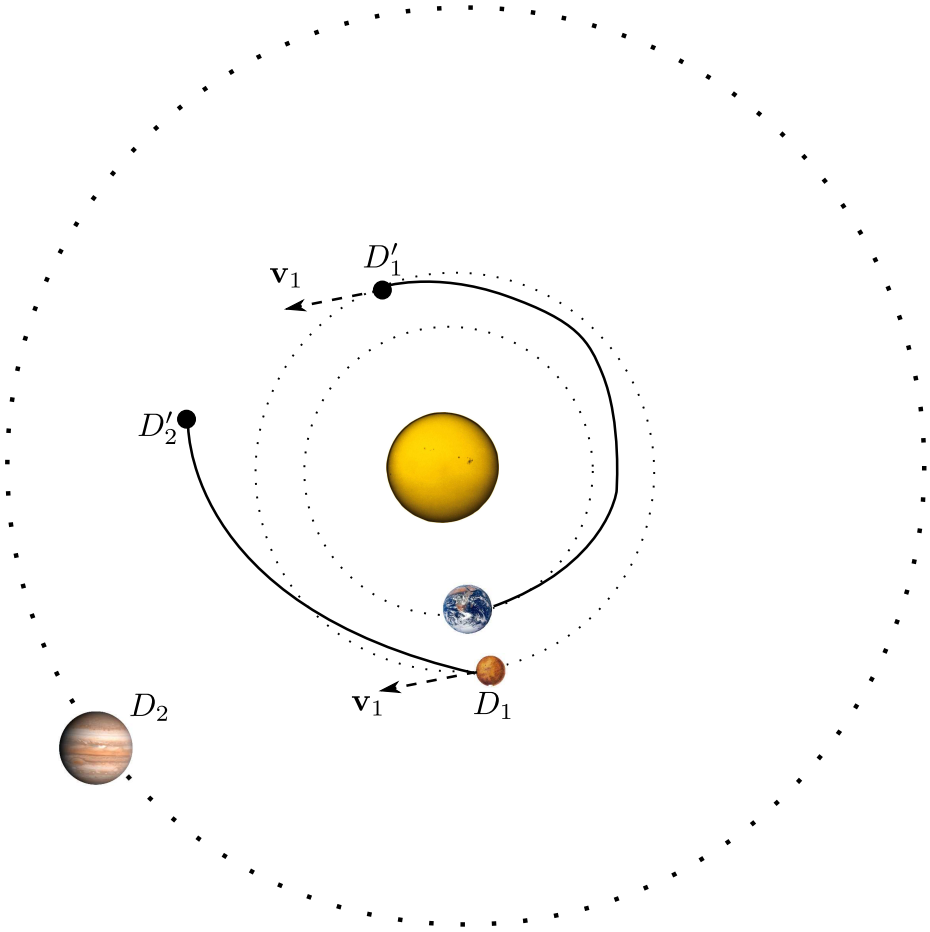


Figure 6.8: Example of a Non-Computable Flyby Using FRA

6.6 Multiple Gravity Assist - Low Thrust (MGA-LT) via the Sims-Flanagan Method

After unsuccessful attempts to model continuous-thrust, multiple flyby trajectories using first the inverse-polynomial shape-based method and then feasible region analysis, the Sims-Flanagan transcription was applied. The Sims-Flanagan transcription is a widely used method in which the continuous-thrust trajectory is discretized into many small time steps, and the thrust applied during each time step is approximated as a small impulse placed at the center of the time step. The trajectory is propagated between control points by solving Kepler's problem [24]. The Sims-Flanagan transcription, when used with an NLP solver such as SNOPT and a suitable initial guess, is very fast and robust. It is used in existing software packages such as GALLOP [33], MALTO [64], and PaGMO [65]. The Sims-Flanagan method was not initially considered as an inner-loop problem solver because moving from continuous integration of the equations of motion to approximating the continuous-thrust trajectory by a set of small impulses is a source of potentially significant error and it was therefore desirable to consider methods involving integration first. However, when those methods failed, Sims-Flanagan was the best remaining choice and, if a reasonable number of time-steps are used, is a good approximation.

The Sims-Flanagan model is combined with a very simple flyby model to create the Multiple Gravity Assist with Low-Thrust (MGA-LT) method. This method has significant advantages. Unlike DTNLP, this transcription avoids the need for numerical integration and results in significant reduction in computing cost in exchange for some lost accuracy. Unlike the Inverse-Polynomial Shape-Based Method and Feasible Region Analysis, MGA-LT handles flybys very effectively.

The MGA-LT algorithm is very easy to formulate. First, the optimizer chooses the components of an impulse vector at the center of each time step. A nonlinear

constraint is applied at each time step to ensure that each impulse ΔV_i is no greater than the maximum Δv that could be achieved if the engine were operated continuously for the length of the time step:

$$\Delta V_i \leq \frac{D n_T T_{\max} (t_f - t_0)}{mN} \tag{6.42}$$

where D is the thruster duty cycle, n_T is the number of thrusters, t_0 and t_f are the beginning and ending times of the time step, m is the mass of the spacecraft at the center of the time step, and N is the number of time steps in the phase. The spacecraft's mass is propagated via Tsiolkovsky's rocket equation [55]:

$$m_{i+1} = m_i \exp\left(-\Delta V_i / g_0 I sp\right) \tag{6.43}$$

where g_0 is the standard Earth gravity at sea level 9.80665 m/s^2 and Isp is the specific impulse of the thruster.

In each phase, the optimizer chooses the initial and final state vectors. The trajectory is then propagated forward from the phase starting location (i.e. the first planet in the phase) and backward from the phase ending location to the mid-point of the phase, known as the "match point." A nonlinear constraint is applied to ensure that all seven components of the state vector are continuous at the match point:

$$\mathbf{s}_{mf} - \mathbf{s}_{mb} = \begin{bmatrix} \Delta x & \Delta y & \Delta z & \Delta v_x & \Delta v_y & \Delta v_z & \Delta m \end{bmatrix} = \mathbf{0} \tag{6.44}$$

where $\mathbf{s}_{mf}$ is the forward propagated state vector and $\mathbf{s}_{mb}$ is the backward propagated state vector, at the match point. Figure 6.9 is a diagram of a phase using the Sims-Flanagan transcription.

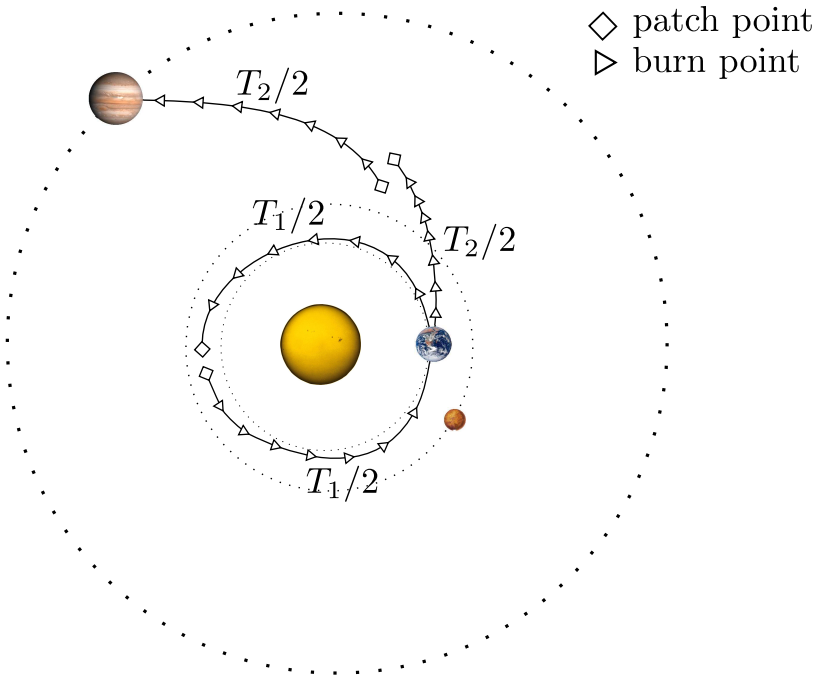


Figure 6.9: Diagram of a Two-Phase Mission Using the Sims-Flanagan Transcription (patch points left discontinuous for illustration purposes)

Two additional nonlinear constraints are imposed at each planetary flyby. As noted in the previous section, the optimizer chooses the vector components of the incoming and outgoing velocity asymptotes from each intermediate destination. Two constraints are therefore necessary to ensure that each flyby maneuver is feasible.

First, it is necessary that the magnitudes of the incoming and outgoing velocity asymptotes are equal:

$$v_{\infty/in} = v_{\infty/out} \tag{6.45}$$

Second, the periapse of the flyby may not be inside the planet. The periapse radius of the flyby is expressed as a function of the gravitational parameter μ_{planet}, the velocity asymptote magnitude v_∞, the radius of the planet r_{planet} and the angle

between the incoming and outgoing velocity asymptotes δ:

$$\frac{r_p}{r_{pl}} = \frac{\mu_{pl}}{v_\infty^2 r_{pl}} \left(\frac{1}{\sin(\delta/2)} - 1 \right) \geq 1.05 \tag{6.46}$$

i.e., the spacecraft is constrained to fly no lower than an altitude of 5% of the planet's radius.

Finally, the total flight time may be constrained. This is done using an explicit linear constraint:

$$\sum_{i=1}^{n_{\text{phases}}} (T_i) - T_{flight-max} \leq 0 \tag{6.47}$$

6.7 Summary of Continuous-Thrust Trajectory Modeling Research

To summarize the process of finding a satisfactory solution method described in the previous sections, based on previous success by researchers at the University of Illinois using Direct Transcription with Nonlinear Programming (DTNLP) [21], we first assumed that the most effective method would be DTNLP coupled with an initial guess obtained via one of two methods: a shape-based method or Feasible Region Analysis (FRA). However, none of these methods proved capable of modeling multi-phase missions that contain flyby maneuvers. The significant problem is that the velocity of the spacecraft at the end of a phase, i.e. going into a flyby, along with the flyby periapse altitude, determine the velocity of the spacecraft at the beginning of the next phase. The shape-based method was unable to find a suitable velocity vector necessary to set up an appropriate flyby. FRA does not satisfy the terminal constraints for each phase until it converges, and therefore, as shown in Figure 6.8, the velocity entering a flyby is not always meaningful because the trajectory does not always actually reach

the planet where the flyby is performed. The shape-based method and FRA were therefore unsatisfactory as initial guess generators for the inner-loop problem.

This discovery motivated a completely different inner-loop solution strategy using the Sims-Flanagan method [24]. The continuous-thrust trajectory was discretized into a sequence of Keplerian arcs with a small impulse at the center of each arc. The trajectory is propagated backward and forward from each destination and continuity constraints are enforced at the center of each phase. A pair of nonlinear constraints are enforced at each flyby to ensure that the incoming and outgoing velocity asymptotes are of equal magnitude and that the spacecraft does not pass through the planet. This parameterization proved to be highly successful and was adopted as the inner-loop trajectory model for continuous thrust.

6.8 Modeling of Launch Vehicles, Thrusters, and Power Systems

Detailed models of real-world launch vehicles, spacecraft thrusters, and power systems are desirable for many mission design problems. Accordingly, this work includes polynomial fits to tabulated performance data for real-world hardware.

Launch vehicle performance is modeled using polynomial fits to tabulated launch vehicle performance data. Launch vehicle performance curves describing mass to orbit as a function of injection energy ($C3$, or v_∞^2) may be found at the NASA Launch Services Program Performance Web Site [66]. A fifth degree polynomial is fit to each launch vehicle performance graph, and then evaluated at whatever injection energy is chosen by the inner-loop problem solver.

Several electric thrusters are modeled in this work. Users may select from the NSTAR, XIPS-25, BPT-4000, and both the high-thrust and high-Isp tunings of the NEXT thruster. Fourth degree polynomial fits of thrust and mass flow rate versus

input electrical power are available in the literature [67–69].

Two power models are available. The first is a solar power model. The user specifies the power available to the thrusters at a distance of $1AU$ from the sun, and at each time step the current available power is given by,

$$P(r) = \frac{P_{1AU}}{r^2} \tag{6.48}$$

Alternatively, the user may specify a fixed electric power. This option represents a nuclear or radioisotope power source.

6.9 Optimization Methods for the Continuous-Thrust Trajectory Optimization Problem

Continuous-thrust trajectories are parameterized by a large number of decision variables and must satisfy constraints on the amount of control (thrust) available at each time step. The resulting continuous-thrust optimization problem has many more decision variables and constraints than an MGA or MGA-DSM problem. For example, an MGA-DSM problem with four flybys and a time of flight constraint is parameterized by 22 decision variables and is subject to 9 constraints. By comparison, an MGA-LT problem with four flybys and ten time steps per phase is parameterized by 167 decision variables and is subject to 39 constraints. If more precision and therefore more time steps are desired, then the problem becomes even larger. It is therefore necessary to use an optimization method that can effectively handle a large number of variables and constraints. Three methods are applied to this problem: Differential Evolution (DE), Particle Swarm Optimization (PSO), and Monotonic Basin Hopping.

6.9.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) were first applied to the MGA-LT problem. As described in Section 4.2, Differential Evolution (DE) and Particle Swarm Optimization (PSO) were shown to be effective for MGA-DSM problems while a Genetic Algorithm (GA) was not. Therefore only DE and PSO were considered as inner-loop problem solvers for continuous-thrust. However, both DE and PSO are designed to optimize a single objective function with no constraints. A penalty function was therefore created that encompasses both the true objective (maximum propellant at end of mission) and the constraints:

$$J = \begin{cases} \sqrt{\sum {c_i}^2} & if \quad \sqrt{\sum c_i} > 0 \\ -m_{final} & if \quad \sqrt{\sum c_i} = 0 \end{cases} \tag{6.49}$$

Here the c_i are the amounts by which each constraint is violated, and so $\sqrt{\sum c_i}$ is the 2-norm of the vector of constraints. J takes a positive value if the constraints are not satisfied, and a negative value if the constraints are satisfied. The EA should therefore first drive the constraints to zero and then maximize the amount of propellant remaining at the end of the mission.

The penalty method of Equation 6.49 was used with both DE and PSO. Unfortunately neither EA was able to consistently find feasible solutions even to simple problems with no flybys. Even when feasible solutions were found, they often used unnecessarily large amounts of propellant. This result is consistent with that of Yam *et al.* [34], who found that the combination of an EA and a penalty method was not a reliable solver for continuous-thrust problems using the Sims-Flanagan transcription. The poor performance of the EAs is likely due to the large size of the problem. EAs perform very well on problems with a small number of decision variables, as in the case of MGA-DSM. However, when the number of decision variables grows from 10-30

to 100-300, EAs can become trapped in local minima. This explains the phenomenon of the EA failing to find a feasible solution - when a penalty method is used, the EA can become trapped at a local minimum of the penalty function that is not in the neighborhood of a feasible, let alone optimal, solution.

6.9.2 Nonlinear Programming

The continuous-thrust Sims-Flanagan problem may be formulated as a Nonlinear Programming (NLP) problem. NLP problems explicitly include constraints and therefore a penalty function is not necessary. Instead, the optimizer solves a problem of the form:

$$\begin{aligned} &\text{Minimize } f(\mathbf{x}) \\ &\text{Subject to:} \\ &\mathbf{x}_{lb} \leq \mathbf{x} \leq \mathbf{x}_{ub} \\ &\mathbf{c}(\mathbf{x}) \leq \mathbf{0} \\ &A\mathbf{x} \leq \mathbf{0} \end{aligned} \tag{6.50}$$

A large scale nonlinear programming solver is therefore necessary. In addition, the EMTG problem is sparse; the thrust magnitude constraint for each time step is affected only by the three vector components defining the thrust for that time step. The problem Jacobian is therefore mostly composed of zero-valued entries. Sparse Nonlinear Optimizer, or SNOPT [70], is therefore a good choice of solver.

SNOPT is capable of solving the continuous-thrust inner-loop problem but has two significant limitations. First, SNOPT performs best if all decision variables, constraints, and the objective function are scaled to the same order of magnitude. Accordingly, for continuous-thrust problems the objective function is not simply to maximize final mass, but rather to minimize:

$$f(\mathbf{x}) = \frac{-m_{final}}{m_{max}} \tag{6.51}$$

where m_{final} is the final mass of the spacecraft and m_{max} is the user defined maximum mass of the spacecraft. The resulting function is always in the range $[-1, 0]$. Similarly, all decision variables are scaled to the range $[0, 1]$ by the transformation:

$$x_{scaled,i} = \frac{x_i - x_{lb,i}}{x_{ub,i} - x_{lb,i}} \tag{6.52}$$

In addition, the constraints are formulated to be of order of magnitude 1. This is done by dividing position patch point constraints by one AU, velocity patch point constraints by 1 AU/TU, and mass patch point constraints by a user-defined "maximum mass" scaling factor that is chosen to represent the maximum capacity of the launch vehicle. Here 1 AU is equal to the semi-major axis of the Earth's orbit and 1 TU is the period of the Earth's orbit divided by 2π.

The second significant limitation of SNOPT is that, like all NLP solvers, it requires an initial guess. There are many types of initial guesses that can be used for continuous-thrust optimization, including constant-thrust propagation [71], ballistic (Lambert) methods [25], shape-based methods (previously discussed in Section 6.4) [26–28] or Feasible Region Analysis (FRA) (previously discussed in Section 6.5) [41, 45]. However, constant-thrust propagation and ballistic initial guesses do not fully explore the possible trajectories that a continuous-thrust spacecraft can follow, and shape-based methods and FRA have already been considered and rejected for the multiple-flyby, continuous thrust problem. It was decided to try a new method for generating an initial guess for the NLP solver, Monotonic Basin Hopping (MBH) [34].

6.9.3 Monotonic Basin Hopping

Monotonic Basin Hopping (MBH) [72] is an algorithm for finding globally optimal solutions to problems with many local optima. MBH works on the principle that many real-world problems have a structure where individual local optima, or "basins" tend to cluster together into "funnels" where one local optimum is better than the rest. A problem may have several such funnels. MBH was originally developed to solve molecular conformation problems in computational chemistry, but has been demonstrated to be effective on various types of interplanetary trajectory problems [34, 73, 74].

First, an initial point $\mathbf{x}$ is randomly chosen. The NLP solver is run using $\mathbf{x}$ as the initial guess. If the NLP solver finds a feasible solution, then that new point $\mathbf{x}^*$ is adopted as the new current point. If the NLP solver does not find a feasible solution, then a new random point is chosen. Once a feasible solution is found, MBH will attempt to "hop" from the feasible and locally optimal $\mathbf{x}^*$ to a better point. This is a two step process: first a small random perturbation vector is added to $\mathbf{x}^*$, producing a new $\mathbf{x}'$, and then the NLP solver is run. If the resulting solution is both feasible and superior to $\mathbf{x}^*$, then it is adopted as the new $\mathbf{x}^*$ and the hopping process begins again. Otherwise, MBH attempts a new hop from the current $\mathbf{x}^*$ and an "impatience" counter $N_{\text{not improve}}$ is incremented. If $N_{\text{not improve}}$ exceeds a user-defined threshold value $\text{Max}_{\text{not improve}}$, then MBH resets and generates a new random $\mathbf{x}$. Each feasible solution is stored in an archive.

The global reset and local hop operators allow MBH the capability both to *explore* the entire solution space and also to *exploit* the local minima in the current "funnel." That is, MBH will explore the solution space via the global reset operator until a feasible solution is found. MBH will then exploit the local funnel, i.e. search for a better local minimum in the vicinity of the current local minimum. The exploitation process continues until the algorithm fails to improve the current point $\text{Max}_{\text{not improve}}$

times, and then MBH will switch back to exploring the entire solution space.

MBH is run until either a specified number of iterations (trial points attempted) or a maximum CPU time is reached, at which point the best solution stored in the archive is returned as the solution to the outer-loop. MBH has three parameters - the stopping criterion, the parameter $\mathrm{N}_{\text{not improve}}$, and the type of random step used to generate the perturbed points $\mathbf{x}'$. In the standard version of MBH, the random step is drawn from a uniform probability distribution in $[-\sigma, \sigma]$. In this work, the value of $\mathrm{N}_{\text{not improve}}$ is set to 50 and σ is set to 0.1. These values were chosen by experiment and work well. MBH is run for up to 30 minutes or 10000 trial points if no feasible solution is found, or up to 8 hours or 10000 trial points if a feasible solution is found. The pseudocode for Monotonic Basin Hopping is listed in Algorithm 6.1.

Algorithm 6.1 Monotonic Basin Hopping (MBH)

```
while not hit stop criterion do
    N_not improve = 0
    generate random point x
    run NLP solver to find point x* using initial guess x
    if (x* is a feasible point) then
        x_current = x*
        while N_not improve < Max_not improve do
            generate x' by randomly perturbing x_current
            run NLP solver to find point x* using initial guess x'
            if x* is feasible and f(x*) < f(x_current) then
                x_current = x*
                N_not improve = 0
            else
                increment N_not improve
            end if
        end while
        save x* to archive
    end if
end while
return best x* in archive
```

6.9.4 Monotonic Basin Hopping with Cauchy Hops

In MBH, candidate points $\mathbf{x}'$ are generated by adding a random number to each element of the best current vector $\mathbf{x}_{current}$. In standard MBH, these random numbers are chosen from a uniform distribution in $[-\sigma, \sigma]$. The value σ can be chosen to be larger if the user desires the "hop" operator to search a larger area of space and smaller if a smaller area is desired. The larger search could be considered a "regional" search and the smaller a "local" search.

In the problems addressed in this work, the very small steps are best used to find slightly better solutions in the immediate vicinity of the current solution. This is valuable because, in a continuous-thrust trajectory optimization, most of the decision variables are thrust control parameters. Each thrust control parameter only affects a small portion of the problem, that is, the current section of the trajectory. For a given pair of end points, there are many combinations of thrust control parameters that will satisfy the trajectory continuity constraints and therefore be feasible. However each choice of the thrust control parameters may result in a slightly different value of the objective function. Therefore it is very easy for the NLP solver to converge to a locally optimal solution and be unaware that there is a better solution in very close proximity. On the other hand, the larger "regional" search ability is useful to find trajectories that are significantly different from the current solution but not quite so different as to require a global reset of MBH. By using a fixed maximum step size σ, standard MBH may be good at either local or regional search, but not both. The value of σ, 0.1, chosen in this work is intended to provide balance between these two search regimes but may not be effective when a very small local search is needed, because the uniform random steps may be too large.

It is therefore desirable to modify the MBH "hop" operator to be effective at both regional and local search. We have drawn inspiration from the world's most effective search agents: foraging and hunting animals. In nature, hunting animals often follow

random search patterns that are mainly made up of very small steps and occasionally take much larger steps. This pattern, which meets the criterion of including both local and regional search steps, can be fit to a Lévy flight distribution; one which has "fat tails" [75]. In a stable distribution, most sample points occur within a narrow range, but in some small fraction of the time very wild behavior can occur. For example, a shark might spend a great deal of time searching one small area for prey, but occasionally will swim a much longer distance in a short time to find new food.

In this work, the standard Cauchy distribution, the simplest of the Lévy flight distributions to implement, is used. The use of a standard Cauchy distribution means that most random steps will be very small, but occasionally the algorithm could choose a large step that could span the entire decision space. The probability density function (PDF) of the standard Cauchy distribution is:

$$f(\rho) = \frac{1}{\pi(1+\rho^2)} \tag{6.53}$$

where ρ is a random number chosen from uniform$[-1, 1]$. Figure 6.10 displays $f(\rho)$. After each random step is added to $\mathbf{x}$ to produce $\mathbf{x}'$, it is possible that $\mathbf{x}'$ might have some elements that are out of bounds. If that is the case, $\mathbf{x}'$ is clipped to the bounding box.

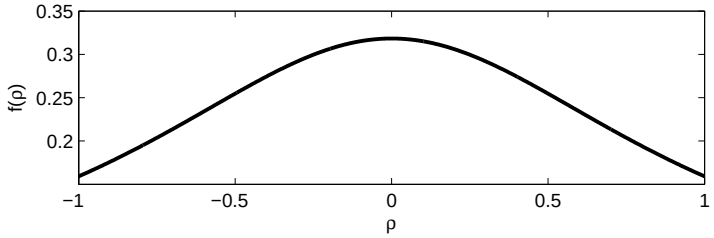


Figure 6.10: Probability density function for a two-sided Cauchy distribution

A thorough statistical analysis of the performance of Monotonic Basin Hopping with Cauchy hops (MBH-C) has not been performed. However the replacement of

the uniform probability density function (for the random step that generates the new candidate point) by the Cauchy distribution that mimics the behavior of foraging animals is intuitive and in our experiments MBH-C appears to perform at least as well as standard MBH.

6.9.5 Automated Choice of Inner-Loop Bounds

As in the case of impulsive-thrust trajectories (MGA, MGA-DSM), upper and lower bounds must be chosen for each decision variable in the MGA-LT problem. Some parameters, notably the launch date (and if applicable the stay time between journeys), are chosen *a priori* by the user. All of the other parameters must have bounds chosen semi-intelligently by a set of simple laws, as listed in Table 6.1. Note particularly that the flight time bounds are functions of τ_i, the orbital periods of the bodies which define the endpoints of each phase.

Table 6.1: Automated Choosing of Inner-Loop Bounds for Continuous Thrust, Sims Flanagan Problems

Parameter	Lower Bound	Upper Bound
Launch date	user-defined	user-defined
Stay time between journeys	user-defined	user-defined
RLA	0.0	2π
DLA	user-defined	user-defined
$v_{\infty-launch}$	0.0	user-defined
For each phase:		
Flight time	$\tau/2$ (repeated flyby of same planet)	5τ (repeated flyby of same planet)
	$0.1min\left(\tau_1, \tau_2\right)$	$1.5max\left(\tau_1, \tau_2\right)$ (outermost body has $a < 2AU$)
		$max\left(\tau_1, \tau_2\right)$ (outermost body has $a \geq 2AU$)
	maximum of 600 days	minimum of 1000 days
Mass at end of phase	0.0 kg	user-defined spacecraft maximum mass
For each time step:		
$\Delta V_{x,y,z}$	-3.0 km/s	3.0 km/s
For each non-rendezvous phase:		
$v_{\infty-incoming-x,y,z}$	user-defined (if terminal intercept)	user-defined (if terminal intercept)
	-10.0 km/s (if intermediate phase)	10.0 km/s (if intermediate phase)
For each phase after the first:		
$v_{\infty-outgoing-x,y,z}$	-10.0 km/s	10.0 km/s

Chapter 7

Example Optimized Continuous-Thrust Trajectories

7.1 Introduction

Three example continuous-thrust missions are presented. The first, and simplest, of the examples is a mission to Jupiter using nuclear electric propulsion (NEP), similar to that of NASA's canceled Jupiter Icy Moons Orbiter (JIMO). An NEP mission is an ideal first test because, unlike in solar electric propulsion (SEP), the performance of an NEP system does not degrade as the spacecraft flies farther away from the Sun. The thrust and specific impulse of the engine can therefore be held constant over the length of the mission. The second example is a mission to Mercury, similar to ESA's proposed BepiColombo mission and also using constant thrust and specific impulse. Both examples also use a fixed initial spacecraft mass, which means that the performance of the launch vehicle need not be modeled. These two examples were chosen because solutions already have been published, using fixed flyby sequences and monotonic basin hopping (MBH) [34]. The results of the first two examples can therefore be compared to known solutions. The standard version of MBH was used for both examples because the Cauchy variant had not yet been developed.

The third and final example is a notional "Uranus Explorer" mission that would travel to Uranus using solar electric propulsion (SEP) and a realistic launch vehicle model. Because the performance of the SEP system varies with the distance of the spacecraft from the Sun and the initial mass of the spacecraft varies with the magnitude of the initial Δv_{LV} supplied by the launch vehicle, this problem is much

more complex than the first two examples and provided a more challenging test of the EMTG. Four versions of the Uranus Explorer problem were optimized, for a variety of maximum flight times and arrival maneuvers. The Cauchy variant of MBH (MBH-C) was used in this example.

7.2 Jupiter Icy Moons Orbiter (JIMO)

The first test case was a nuclear electric propulsion (NEP) mission to Jupiter. The Jupiter Icy Moons Orbiter, or JIMO, was a proposed NASA mission to Jupiter that was canceled due to high cost [76–80]. Yam, Di Lorenzo, and Izzo re-examined this problem using the Sims-Flanagan transcription and monotonic basin hopping in 2011 [34]. They used a fixed sequence of flybys, but their work provides a valuable comparison because their solutions are comparable to the solutions that are generated by the inner-loop of the EMTG presented here. In the JIMO problem, the thrust, specific impulse, and initial mass of the spacecraft are fixed. Since this was an early test and predates the invention of the Cauchy variant of monotonic basin hopping, the standard version of MBH was used instead.

To enable benchmarking against the work of Yam, Di Lorenzo, and Izzo, the same problem assumptions used in their study were used here. The exception, of course, is that the flyby sequence was not fixed in this study but was chosen by the automaton. Table 7.1 contains the problem assumptions. A fixed initial mass was used, along with fixed thrust and I_{sp}.

The EMTG ran for 37.4 hours on a 3.06 GHz Intel Core i7, and evaluated 199 solutions, of which 47 were feasible. Table 7.2 shows the top 20 sequences evaluated. The best sequence was found to be Earth-Earth-Earth-Jupiter (EEEJ), which delivered 17569 kg to Jupiter rendezvous out of a starting mass of 20000 kg. Table 7.3 describes the optimal solution, and Figure 7.1 shows the trajectory. The small red lines ema-

Table 7.1: Assumptions for the Jupiter Icy Moons Orbiter (JIMO) Mission

Option	Value
Arrival type	rendezvous (match position and velocity with Jupiter)
Launch window open date	1/1/2020
Launch window close date	9/28/2022
Flight time upper bound	15 years
Propulsion type	fixed I_{sp} and thrust
Thrust (N)	2.26
I_{sp} (s)	6000
Initial mass (kg)	20000
Maximum ΔV_{LV} (km/s)	1.925
Number of time steps per phase	10
Maximum number of flybys	8
GA Population Size	100
Inner-Loop run time per sequence	2 hours

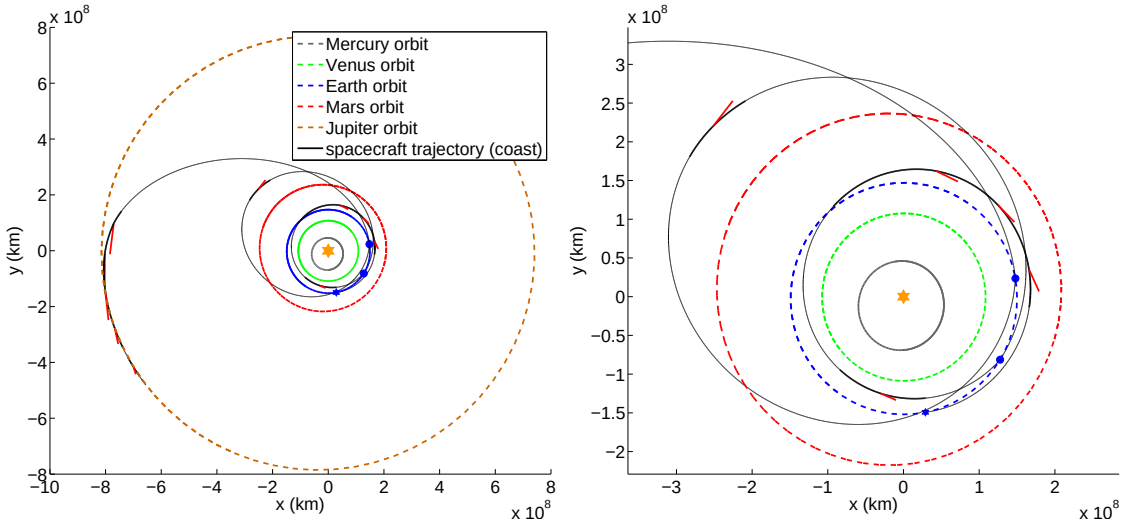
nating from the spacecraft trajectory represent the small impulses that approximate the continuous-thrust when the Sims-Flanagan transcription is used. Furthermore, the path of the spacecraft is represented by a bold line when the thrusters are on and a thin line when they are off. The best final mass of 17569 kg for the EEEJ flyby sequence is slightly better than the value of 17102 kg found by Yam *et al.* [34], but their solution was found for an EEJ sequence. The best EEJ solution found by EMTG delivered only 16775 kg to Jupiter. This difference is possibly because of the maximum ΔV_{LV} of 1.925 km/s used here vs the 2.0 km/s used by Yam *et al.*, but could also be due to the stochastic nature of the MBH solver. Because the solver relies on random choices of starting location, it is possible that the best solution will not be found every time.

Table 7.2: Top 20 Candidate Sequences for the Jupiter Icy Moons Orbiter (JIMO)

Sequence	Final Mass (kg)
EEEJ	17569
EEEEJ	17535
EEEVEEJ	17035
EEMJ	16995
EMEEJ	16825
EEJ	16775
EEVVEJ	16445
EEVEJ	16356
EVEEJ	16344
EVVJ	16266
EEVJ	16234
EVEJ	16203
EMEVVEJ	16066
EMEJ	15952
EMEVEJ	15923
EJ	15905
EVMVJ	15901
EEEVEJ	15897
EJJ	15706
EVMEJ	15583

Table 7.3: Itinerary for the Jupiter Icy Moons Orbiter (JIMO)

Date	Event Type	Location	ΔV / v_∞ (km/s)	altitude (km)	δ (°)	mass (kg)
7/2/2022	launch	Earth	1.925	-	-	20000
8/20/2023	flyby	Earth	7.23478	318.905	84.6955	19491.5
10/1/2025	flyby	Earth	7.60437	318.905	49.1239	19316.3
10/10/2029	rendezvous	Jupiter	0	-	-	17568.6



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 7.1: Trajectory for the Jupiter Icy Moons Orbiter (JIMO)

7.3 BepiColombo

The next example case is the European Space Agency's BepiColombo mission to Mercury [34, 81, 82]. BepiColombo will use low-thrust electric propulsion and flybys of Venus and Mercury to reach its final orbit about Mercury. The specific version of BepiColombo studied here is the original concept study that assumed launch in 2008. This is done to enable comparison with the results of Yam *et al.* [34], who used

monotonic basin hopping with nonlinear programming to optimize an Earth-Venus-Venus-Mercury-Mercury-Mercury (EVVYYY) trajectory and a 2008 launch. As in the JIMO case, the thrust, specific impulse, and initial mass of the spacecraft are fixed. Also as in the JIMO case, since this was an early test, the Cauchy variant of monotonic basin hopping was not used. Table 7.4 describes the problem assumptions.

Table 7.4: Assumptions for the BepiColombo Mission

Option	Value
Arrival type	intercept (match position) with bounded v_∞
Maximum arrival v_∞	0.5 km/s
Launch window open date	8/1/2009
Launch window close date	4/27/2012
Flight time upper bound	15 years
Propulsion type	fixed I_{sp} and thrust
Thrust (N)	0.34
I_{sp} (s)	3200
Initial mass (kg)	1300
Maximum ΔV_{LV} (km/s)	1.925
Number of time steps per phase	10
Maximum number of flybys	8
GA Population Size	100
Inner-Loop run time per sequence	2 hours

The EMTG was run four times on the BepiColombo problem. The best solutions from each of the four runs are shown in Table 7.5. The best sequence found in Run 3 was EVVYYY, the same as used by Yam *et al.*, but with a best final mass of 1070 kg, slightly better than the 1062 kg found by Yam *et al.*. However the best overall sequence was found in Run 1, an EEVVYY sequence with a best final mass of 1112 kg. This illustrates how the automaton can improve on a flyby sequence chosen by a skilled analyst. The top 20 sequences out of 211, evaluated in Run 1, are listed in Table 7.6. Table 7.7 describes the best solution found in Table 7.6, and the corresponding trajectory is shown in Figure 7.2.

Table 7.5: Best Solutions found by Four Runs of the BepiColombo Problem

Rank	Best Sequence	Mass Delivered (kg)	# Sequences Evaluated (feasible)	Run time (days)
1	EEVVYY	1112	211 (62)	11.7
2	EEVVYY	1076	237 (64)	11.4
3	EVVYYY	1070	261 (66)	11.8
4	EVVYY	1062	221 (61)	10.8

Table 7.6: Top 20 Candidate Sequences for the BepiColombo Mission, Run 1

Sequence	Final Mass (kg)
EEVVYY	1112
EVVYY	1077
EEEVVYY	1077
EEVVVYY	1076
EEVYYY	1061
EEVYY	1045
EMVVYY	1038
EEVVY	1030
EEEEVYY	1030
EEEVYY	1026
EEVY	1024
EVVY	1020
EEVEVYY	1013
EVVVY	1006
EVYY	998
EMEVVYY	972
EVYYY	970
EVY	964
EEYYY	937
EMEVY	930

Table 7.7: Itinerary for BepiColombo, Run 1

Date	Event Type	Location	ΔV / v_∞ (km/s)	altitude (km)	δ (°)	mass (kg)
09/26/2011	launch	Earth	1.925	-	-	1300
08/03/2016	flyby	Earth	3.71886	21944.5	51.4687	1272.61
10/07/2017	flyby	Venus	5.16911	3895.12	33.3306	1272.61
02/19/2019	flyby	Venus	6.96256	302.599	45.4456	1272.61
04/10/2021	flyby	Mercury	2.9265	122	51.5511	1159.62
05/23/2022	arrival	Mercury	0.5	-	-	1112.32

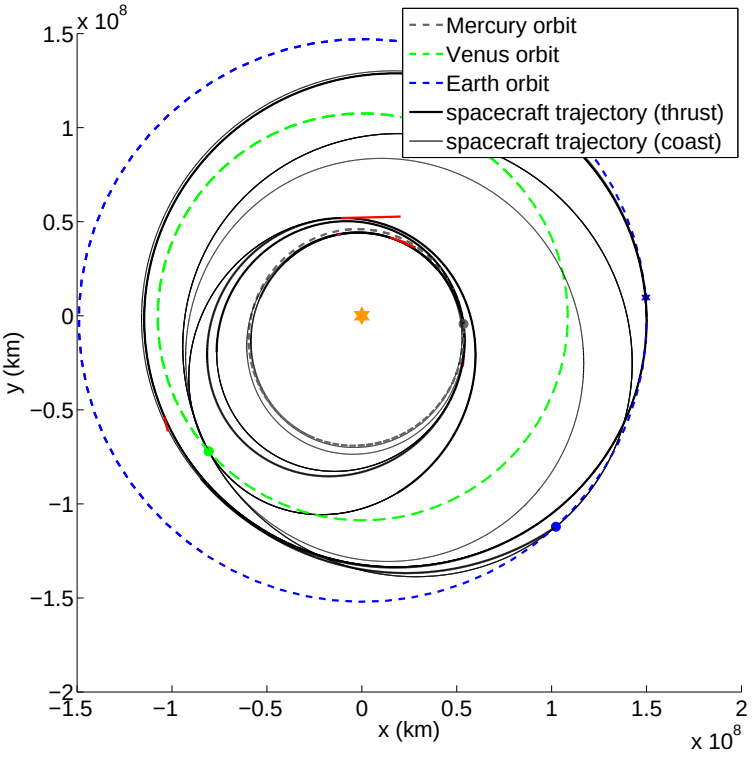


Figure 7.2: Best Trajectory for BepiColombo, Run 1

7.4 Uranus Explorer

Finally, the EMTG-LT was used to study a notional "Uranus Explorer" mission in the 2030-2040 time frame. For this study, a solar electric propulsion system (SEP) was used. The SEP system consists of two NEXT thrusters [69] running in parallel with a duty cycle of 90% and a solar array capable of supplying 25 kW of power at 1 AU. The SEP system has a mass of 1200 kg and is loosely based on a NASA GRC concept study of a multi-purpose SEP stage [60]. The launch vehicle used was the Atlas V 551, the most capable version of Atlas V [66]. The initial mass of the spacecraft is assumed to be equal to the maximum mass that Atlas V 551 can put onto whatever departure trajectory is chosen by the optimizer. Four variants of the Uranus Explorer mission were studied, two ending in an intercept of Uranus with a v_∞ of less than 8 km/s and two ending in an impulsive insertion into a high-eccentricity elliptical orbit about Uranus. This upper bound of 8 km/s was motivated by a Decadal Survey study that considered v_∞ of as high as 7.3 km/s [61]. Flight times of 15 and 20 years were studied for both types of mission. Table 7.8 shows the problem assumptions common to all four variants. Table 7.9 defines the four cases, and Table 7.10 describes the additional parameters used only for Cases 3 and 4. Both cases were run in serial on a 3.06 GHz Intel Core i7. The objective was to maximize the amount of mass delivered to Uranus.

This problem is much more complex than the previous examples because the performance of the SEP system varies with the distance from the Sun and because the initial mass of the spacecraft is not fixed, but rather is dependent on the magnitude of the initial Δv_{LV} chosen by the optimizer. As a result, the "Uranus Explorer" mission is a challenging final test for the EMTG.

Table 7.8: Assumptions Common to all Uranus Explorer variants

Launch window open date	1/1/2030
Launch window close date	12/31/2040
Propulsion type	SEP
Thruster type	2x NEXT
Available power at 1 AU	25 kW
Minimum power for thruster operation	2.504 kW
Thruster duty cycle	90%
Launch Vehicle	Atlas V 551
Number of time steps per phase	40
Maximum number of flybys	8
GA Population Size	200
Inner-Loop run time per sequence	8 hours

Table 7.9: Description of Uranus Explorer Cases

Case Number	Flight time	Arrival Type
1	15 years	Intercept with bounded v_∞
2	20 years	Intercept with bounded v_∞
3	15 years	orbit insertion
4	20 years	orbit insertion

Table 7.10: Additional Parameters for Uranus Explorer Cases 3 and 4

Final orbit a	2310000 km
Final orbit e	0.98
I_{sp} of chemical motor	323 s
Dry mass of electric propulsion stage	1200 kg

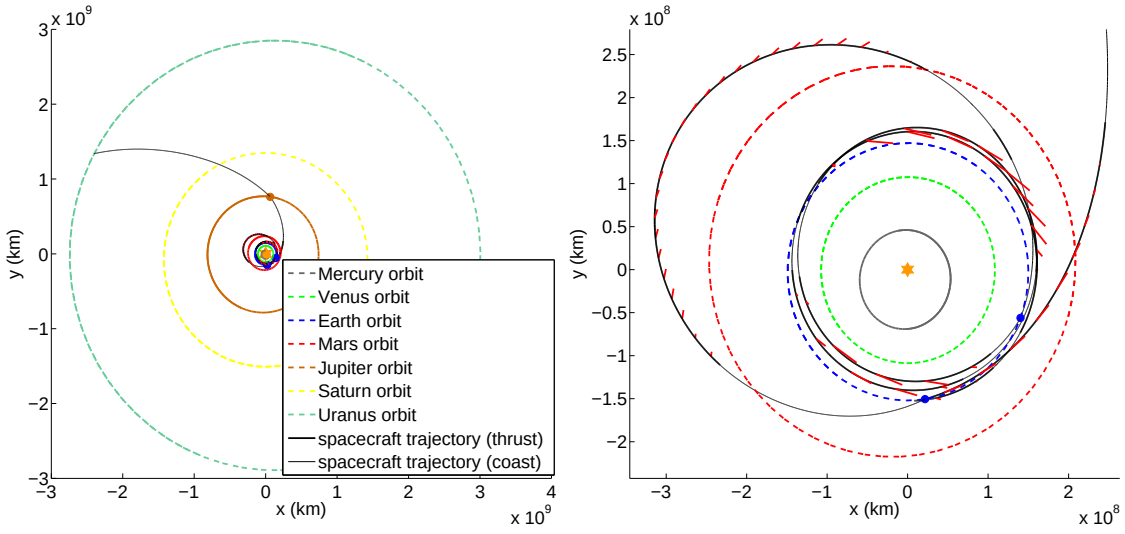
For Case 1, the EMTG ran for 13.2 days and evaluated 455 flyby sequences, of which 18 were feasible with the given assumptions and constraints. The best sequence found was EEEJU and delivered 5972 kg to Uranus. Table 7.11 shows the 18 feasible candidate sequences. Table 7.12 describes the optimal solution, and Figure 7.3 shows the optimal trajectory. Note that most of the thrusting occurs inside the orbit of Mars. This is because there is insufficient solar power available to operate the SEP system, i.e. less than 2.5 kW, for parts of the trajectory beyond approximately 3 AU from the Sun.

Table 7.11: Top 18 Candidate Sequences for the Case 1 Uranus Explorer Mission

Sequence	Final Mass (kg)
EEEJU	5972
EEJU	5508
EVJU	4154
EVVJU	4059
EEU	3879
EMJU	3646
EJU	3542
EMU	2565
EVMJU	2408
EU	2172
EVMU	2108
EEEVMU	1882
EYVEU	1743
EYEU	1737
EEVMU	1436
EVYEU	869
EMYJU	529
EYYVJU	195

Table 7.12: Itinerary for the Case 1 Uranus Explorer Mission

Date	Event Type	Location	ΔV / v_∞ (km/s)	altitude (km)	δ (°)	mass (kg)
5/12/2031	launch	Earth	0.620748	-	-	6568
8/30/2033	flyby	Earth	7.06871	655.668	84.4371	6211
6/29/2035	flyby	Earth	7.42014	318.905	42.5835	6064
1/27/2037	flyby	Jupiter	5.62729	3.19E+06	33.1042	5972
4/3/2046	intercept	Uranus	5.53818	-	-	5972



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 7.3: Trajectory for the Optimal Case 1 Uranus Explorer Mission

For Case 2, which is the same as Case 1 but with a maximum flight time extended from 15 to 20 years, the EMTG ran for 9.2 days and evaluated 511 flyby sequences, of which 41 were feasible. The best sequence found was EMMEJU and delivered 5704 kg to Uranus. Table 7.13 shows the top 20 candidate sequences. Table 7.14 describes the optimal solution, and Figure 7.4 shows the optimal trajectory. Note that in this case the EMTG failed to find a solution as good as Case 1, which is surprising because the best Case 1 trajectory should have been feasible for Case 2. This is because the inner-loop solver failed to find the optimal solution for the EEEJU flyby sequence with 15 year flight time, finding only a best delivery mass of 5513 kg instead of the

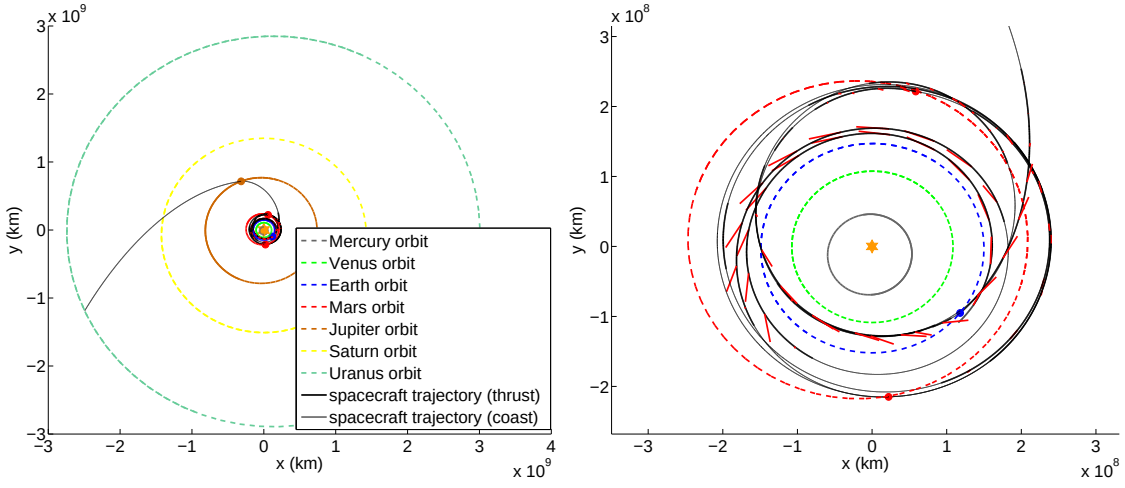
5792 kg found in Case 1. Unfortunately, this is because Monotonic Basin Hopping is a stochastic optimizer and no matter how many times one runs it, there still exists the possibility of not finding the globally optimal solution.

Table 7.13: Top 20 Candidate Sequences for the Case 2 Uranus Explorer Mission

Sequence	Final Mass (kg)
EMMEJU	5704
EEVEJU	5613
EEEJU	5513
EEEU	5495
EEJU	5428
EMMJU	5428
EMEJU	5055
EEMJU	4987
EEVMJU	4712
EESU	4502
EJU	4193
EVMJU	4058
EMMEU	4055
EMEEJU	3964
EMEU	3927
EMVEJU	3904
EEMMEU	3829
EVEJU	3635
EMVJU	3354
EVVEJU	3301

Table 7.14: Itinerary for the Case 2 Uranus Explorer Mission

Date	Event Type	Location	ΔV / v_∞ (km/s)	altitude (km)	δ (°)	mass (kg)
3/9/2038	launch	Earth	0	-	-	6613.27
1/2/2041	flyby	Mars	1.94249	5182.94	29.307	6027.58
7/1/2045	flyby	Mars	3.35453	169.85	44.0079	5952.94
8/13/2047	flyby	Earth	7.48719	318.905	44.5961	5776.57
11/10/2049	flyby	Jupiter	9.9837	598860	115.075	5703.61
11/7/2057	intercept	Uranus	7.20667	-	-	5703.63



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 7.4: Trajectory for the Optimal Case 2 Uranus Explorer Mission

For Case 3, which is similar to Case 1 but ends in an impulsive insertion into orbit about Uranus rather than an intercept, the EMTG ran for 6.0 days and evaluated 405 flyby sequences, of which 15 were feasible. There are fewer feasible sequences for this orbit insertion case than for the previous intercept cases because there is a constraint that the spacecraft must arrive in the vicinity of Uranus with at least the dry mass of the electric propulsion stage (1200 kg). This is because the electric propulsion stage

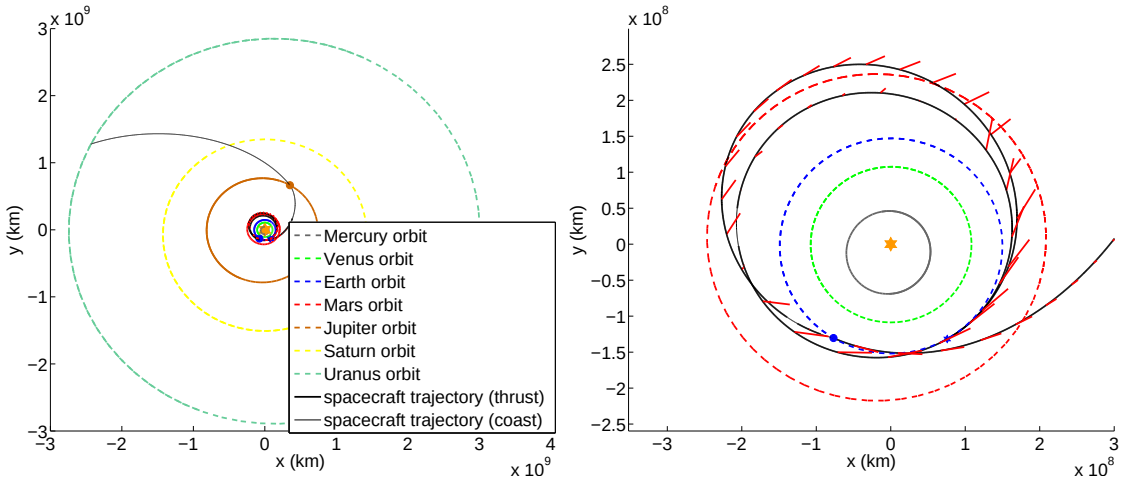
must be jettisoned before the spacecraft performs its impulsive orbit insertion burn. The best sequence found was EEJU and delivered 3260 kg to Uranus. Table 7.15 shows the top 15 candidate sequences. Table 7.16 describes the optimal solution, and Figure 7.5 shows the optimal trajectory.

Table 7.15: Top 15 Candidate Sequences for the Case 3 Uranus Explorer Mission

Sequence	Final Mass After Insertion Burn (kg)
EEJU	3260
EEU	2157
EMEU	1753
EMJU	1721
EJU	1638
EMMJU	1423
EVVJU	1005
EMU	979
EEMU	930
EU	859
EVMMU	833
EEMJU	809
EEEMU	799
EMVJU	725
EVMU	522

Table 7.16: Itinerary for the Case 3 Uranus Explorer Mission

Date	Event Type	Location	ΔV / v_∞ (km/s)	altitude (km)	δ (°)	mass (kg)
7/22/2031	launch	Earth	2.21833	-	-	6051.26
5/19/2034	flyby	Earth	7.56222	318.905	47.2847	5510.67
5/9/2036	flyby	Jupiter	8.19639	1.76E+06	71.4582	5395.57
7/22/2046	insertion	Uranus	4.99964	-	-	3259.56



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 7.5: Trajectory for the Optimal Case 3 Uranus Explorer Mission

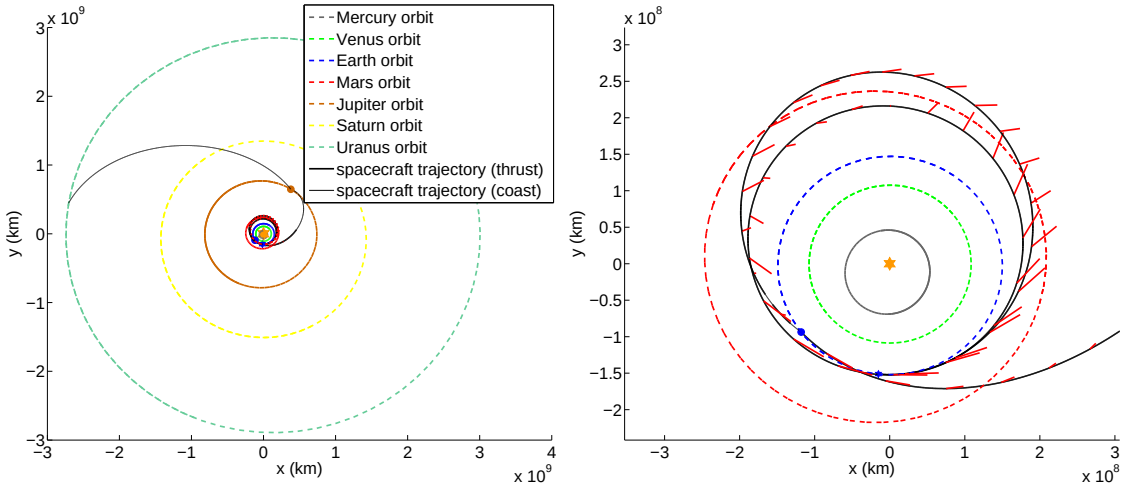
Finally, for Case 4, which is similar to Case 2 but ends in an impulsive insertion into orbit about Uranus rather than an intercept, the EMTG ran for 10.9 days and evaluated 363 flyby sequences, of which 16 were feasible. The best sequence found was EEJU and delivered 3753 kg to Uranus. Table 7.17 shows the top 15 candidate sequences. Table 7.18 describes the optimal solution, and Figure 7.6 shows the optimal trajectory.

Table 7.17: Top 16 Candidate Sequences for the Case 4 Uranus Explorer Mission

Sequence	Final Mass After Insertion Burn (kg)
EEJU	3753
EEEU	3460
EVEJU	3418
EMEJU	3127
EEMEU	2717
EMJU	2701
EEU	2631
EJU	2024
EVJU	2019
EVVJU	1838
EMU	1701
EEMU	1417
EEEMU	1202
EEEVU	560
ESU	292
EVEEEJU	5

Table 7.18: Itinerary for the Case 4 Uranus Explorer Mission

Date	Event Type	Location	ΔV / v_∞ (km/s)	altitude (km)	δ (°)	mass (kg)
6/14/2030	launch	Earth	1.9915	-	-	6156.86
4/27/2033	flyby	Earth	7.60574	318.905	49.1893	5520.43
4/11/2036	flyby	Jupiter	10.846	359216	125.08	5443.91
6/14/2050	insertion	Uranus	3.24026	-	-	3753.11



(a) Full Trajectory

(b) Inner-Planet Flybys Only

Figure 7.6: Trajectory for the Optimal Case 4 Uranus Explorer Mission

Chapter 8

Conclusions

8.1 Summary

Historically, the preliminary design of interplanetary trajectories has been an expensive and time-consuming process, requiring a team of analysts. There are eight planets in the solar system, each of which is a potentially useful flyby target. In a five to twenty year mission, there is time for the spacecraft to perform a lengthy sequence of flybys. But how many flybys should be performed, and of what planets? Tens of millions of possible sequences exist. Most of these sequences are clearly infeasible, i.e. there is no reason to go to Neptune on the way from Earth to Venus. An experienced team of analysts can, with much work, find good candidate sequences as was done for historical missions such as Galileo. However, this is an expensive and time consuming process.

The objective of this work was to develop a mission design automaton that can find both the best flyby sequence and also the optimal trajectory given only the basic requirements of the mission, i.e. the launch date, the upper bound on the flight time, and the starting and destination planets. Total enumeration of the possible flyby sequences is impractical because there are tens of millions of possibilities and hundreds of them may be feasible. Instead, it is necessary to develop a method that can, with strong confidence, find both the number of flybys and the identities of the flyby planets while evaluating only a small fraction of the available candidate sequences.

In this work, the interplanetary mission design problem is converted into a hybrid optimal control problem (HOCP) with two nested optimization loops. The outer-loop is an integer programming problem to determine the flyby sequence. A Genetic Algorithm (GA) is used to solve the outer-loop problem. By associating a cost metric with each candidate sequence, it can identify characteristics such as poorly performing subsequences, e.g. Earth-Jupiter-Earth or Earth-Neptune-Mercury, and rapidly discard them. The inner-loop is a continuous trajectory optimization problem to find the optimal trajectory for a given candidate flyby sequence chosen by the outer-loop.

A number of technologies needed to be developed in order to build a HOCP mission design automaton. First, a GA needed to be developed that can solve an integer programming problem with a variable length decision chromosome, as in the case of the flyby sequence problem. Conventional GAs are designed to evolve a decision chromosome of fixed length. Second, trajectory models needed to be chosen for both the impulsive-thrust and continuous-thrust trajectory optimization problems. Because the HOCP will evaluate many candidate solutions, the trajectory models needed to be computationally inexpensive so that they may be evaluated very quickly. Third, efficient and satisfactory inner-loop problem solvers needed to be found for both impulsive-thrust and continuous-thrust trajectory optimization problems. These solvers needed to be perfectly robust - they must always return a cost function value either representing the Δv or propellant cost of the trajectory, or alternatively a metric of infeasibility for flyby sequences where no feasible trajectory can be found. Finally, upper and lower bounds must be chosen for each of the decision variables. These choices are normally made by an experienced analyst, but in a HOCP automaton the trajectory optimization problem is generated in real-time by the outer-loop and therefore there is no opportunity for a human to make any choices. Instead, the upper and lower bounds must be chosen autonomously.

In this work, a novel "null gene" transcription method was developed that allows

the outer-loop GA to choose a variable-length sequence of flybys. Each planet in the solar system is assigned an integer code, and number of integer codes are also chosen to represent "no flyby." This is very similar to the concept of "null genes" in biological evolution, where some genes do not represent physical traits. The GA may control the length of the flyby sequence by assigning such "null genes" to elements of the decision chromosome.

The existing Multiple Gravity Assist (MGA) and Multiple Gravity Assist with one Deep Space Maneuver (MGA-DSM) trajectory models were chosen to represent impulsive-thrust trajectories [14, 15]. In both models, the spacecraft is assumed to follow a Keplerian trajectory. In the MGA model, the spacecraft may perform impulsive maneuvers only at the periapse of each flyby. In the MGA-DSM model, a single impulse may occur anywhere between each pair of planets, an option that in some cases may be used to reduce the total cost of a mission. Both models parameterize impulsive-thrust trajectories with a minimal number of decision variables and therefore may be evaluated very quickly. However, the existing MGA and MGA-DSM models suffered from a significant flaw that had to be corrected in order to make them suitable for a HOCP automaton: in flyby sequences where the outer-loop problem solver chose the first flyby in the sequence to be of the same planet from which the spacecraft launched, the inner-loop problem solver frequently chose a trajectory in which the spacecraft never actually left the sphere of influence of the first planet and so its approach to the planet for the first flyby was not hyperbolic. In this situation, the equations governing the flyby were not valid and unpredictable, non-physical results were observed. It was therefore necessary to introduce a penalty function to force every flyby to be hyperbolic. Several penalty functions were considered, and eventually an effective method was found that penalizes cases where the orbital energy of the spacecraft at the boundary of the planet's sphere of influence is negative or near zero, which would signify a non-hyperbolic flyby. With this addition, the MGA

and MGA-DSM models became suitable for the inner-loop of the HOCP automaton.

Several trajectory models were considered for the continuous-thrust inner-loop problem. Due to the success experienced by many previous researchers, the first method considered was Direct Transcription with Nonlinear Programming (DTNLP), in which the continuous trajectory is discretized into a sequence of time steps. The state vector of the spacecraft is chosen by a Nonlinear Programming (NLP) problem solver at the left-hand boundary of each time-step, and the control variables (thrust control vector components) are chosen at the left-hand boundary and a set of interior points within each time-step. The equations of motion are integrated across each time-step using an implicit Runge-Kutta method and nonlinear defect constraints are enforced at the boundaries of each time step to ensure a continuous trajectory. However, this method requires a good initial guess of the state and control variables. Two methods were considered as candidates to produce this initial guess.

The first initial guess generator considered in this work is a shape-based method originally developed by Wall and Conway [27], and later extended by Wall *et al.* [62, 63], in which the continuous-thrust trajectory is assumed to take a certain shape that can be parameterized by a small number of values and has endpoints at the desired starting and ending bodies of the continuous-thrust arc. The second method is feasible region analysis, as developed by Chilan and Conway [41,45]. FRA assures that the inner-loop always returns either a near-optimal trajectory or a measurement of how close the spacecraft can get to the destination if the transfer cannot be completed. Unfortunately it was found that neither method can effectively generate a trajectory that includes flybys.

After both the shape-based method and FRA were shown to be impractical as initial guess generators for a DTNLP approach to the multiple-flyby continuous-thrust trajectory optimization problem, a completely new approach was developed. Instead of DTNLP, the continuous-thrust trajectory was modeled using the Sims-Flanagan

transcription [24], in which the continuous-thrust trajectory is discretized into a set of Keplerian arcs and small impulses are placed in the center of each arc to approximate the continuous-thrust propulsion. The trajectory is propagated forward and backward from the end-points of each phase and a set of nonlinear continuity constraints are enforced at the center of each phase. A pair of nonlinear constraints are enforced at each flyby to ensure that the incoming and outgoing velocity asymptotes are of equal magnitude and that the spacecraft does not pass through the planet. This parameterization, known as Multiple Gravity Assist with Low Thrust, or MGA-LT, proved to be highly successful and was adopted as the inner-loop trajectory model for continuous thrust.

Several real-valued optimization methods were considered as inner-loop problem solvers for the HOCP mission design automaton. The impulsive-thrust trajectory models considered in this work, MGA and MGA-DSM, are both characterized by a small number of decision variables and very few constraints. Both are therefore suitable for optimization via evolutionary algorithms (EAs). Penalty functions were used to enforce the few constraints (i.e. total flight time and flyby feasibility). Two EAs, Differential Evolution (DE) and Particle Swarm Optimization (DE) were initially tested. Both DE and PSO were found to be very satisfactory solvers for MGA-DSM trajectories, but both tended to prematurely converge to local minima when applied to MGA trajectories. However, a cooperative algorithm that rapidly switches back and forth between DE and PSO, known as COOP, was found to be a very robust optimizer for MGA trajectories.

In contrast to impulsive-thrust trajectories, continuous-thrust trajectories modeled using the Sims-Flanagan method are characterized by hundreds of decision variables and tens of constraints. It is impractical to aggregate these constraints into a penalty function. Therefore instead of an EA, an NLP problem solver is used to optimize the continuous-thrust trajectories in this work. However, all NLP problem

solvers require an initial guess. Unfortunately the two most promising choices of initial guess generator, the shape-based method of Wall *et al.* and FRA, had already been shown to be impractical for multiple-flyby continuous-thrust trajectories. Instead, the Monotonic Basin Hopping (MBH) search heuristic was used. In MBH, the NLP problem solver is run repeatedly from a completely random initial guess until a feasible solution is found. Then, small uniform random "hop" values are added to the converged decision vector and the NLP problem solver is run again. If a solution with a better cost is found, then the new point is adopted and the process continues. Otherwise a new random hop is performed from the original feasible solution and the NLP problem solver is run again. This process continues until either a specified number of iterations or a specified time is reached. MBH was found to be a very satisfactory inner-loop problem solver and was incorporated into the HOCP mission design automaton. Later, a modification to MBH was introduced in which the uniform random "hop" perturbation was replaced by a perturbation drawn from a Cauchy distribution. This method, known as Monotonic Basin Hopping with Cauchy hops, or MBH-C, was inspired by the foraging patterns of animals. MBH-C has not been exhaustively tested but appears to perform at least as well as the standard MBH.

Finally, a set of rules were developed to determine upper and lower bounds for each decision variable in the inner-loop trajectory optimization problem. Specific rules were developed for bounding flyby altitudes, thrust vectors, terminal velocity vectors, and flight times. These rules allowed the inner-loop problem solvers to operate without any *a priori* knowledge of the trajectory optimization problem being solved, an essential characteristic of a HOCP mission design automaton.

The HOCP mission design automaton was demonstrated to solve a number of example mission design problems. Two impulsive-thrust cases were optimized using the MGA trajectory model, and the COOP evolutionary solver. The first such case

was a Galileo-like mission to Jupiter and the second was a Cassini-like mission to Saturn. In both cases, the HOCP automaton found both the optimal flyby sequence and the optimal trajectory with no *a priori* information about the problem except a range of acceptable launch years, a maximum flight time, and the parameters of the desired final orbit about the destination planet. In the Saturn case, the automaton found the same flyby sequence used by the actual Cassini mission.

Next, the HOCP mission design automaton was used to design a Cassini-like mission to Saturn using the MGA-DSM model, i.e. a mission in which an impulsive-thrust maneuver could be applied at any point between each pair of planets. The HOCP automaton found not only the same flyby sequence as the actual Cassini mission but also the same trajectory, i.e. there were only small differences between the dates of the flybys and the magnitude of the impulsive-thrust maneuvers in the solution generated by the automaton vs the dates and maneuvers used by the actual Cassini mission. This is a very significant achievement because while the original Cassini trajectory design required the work of a team of experienced analysts, the HOCP mission design automaton found the same solution using only very basic *a priori* information and two days of computing time on a personal computer.

Finally, the HOCP mission design automaton was used to optimize three continuous-thrust examples using the MGA-LT model. The first two example cases were a mission to Jupiter, similar to the Jupiter Icy Moons Orbiter (JIMO), and a mission to Mercury that is very similar to the BepiColombo mission. The spacecraft in both examples had a constant thrust and specific impulse, and no launch vehicle model was used. The standard form of MBH was used with an NLP problem solver to solve the inner-loop trajectory optimization problem. In both cases, the HOCP automaton found a flyby sequence and continuous-thrust trajectory that were superior to, i.e. delivered more mass to the destination, than the published solutions to these problems. In both cases, the superior cost value was a result of the HOCP choosing

a better flyby sequence than that chosen by human analysts in the published solutions. The last example problem solved using the automaton was a notional "Uranus Explorer" mission. In this example, an existing launch vehicle (the Atlas V-551 with a Star-48 upper stage), an existing electric thruster (NEXT), and a realistic 25 kW solar power supply were used. This problem is especially challenging because the solar-electric propulsion system may only be used within a distance of 3 AU from the Sun, after which there is insufficient power to operate the thruster and the spacecraft must coast to its destination. The MBH-C heuristic combined with an NLP problem solver was used to solve the inner-loop trajectory optimization problem. The "Uranus Explorer" example was therefore a test of all of the advanced capabilities of the continuous-thrust HOCP mission design automaton. Four sub-cases were examined, two with a 15 year upper bound on the flight time and two with a 20 year upper bound. The terminal maneuver of one of each length of mission was an intercept of Uranus with bounded v_∞ and the other was an impulsive-thrust insertion into a highly elliptical orbit about Uranus. In all four cases, the automaton found a flyby sequence and continuous-thrust trajectory delivering thousands of kilograms to Uranus, showing that the automaton is capable of solving very complex continuous-thrust problems that incorporate variable thrust and specific impulse, an accurate launch vehicle model, and a destination far enough from the Sun that the electric thruster may only be used for part of the trajectory.

The technologies developed in this work were integrated into a software package, the Evolutionary Mission Trajectory Generator (EMTG). Three versions of the EMTG were developed, for MGA, MGA-DSM, and MGA-LT trajectories. The EMTG is a HOCP mission design automaton capable of optimizing multiple-flyby trajectories with only very basic *a priori* information about the mission. The EMTG replaces man-years of work by an team of experienced analysts with a few days of run-time on a personal computer. To date, the EMTG is the only known tool that

can choose both the optimal flyby sequence and the optimal trajectory for missions using impulsive or continuous-thrust propulsion and very few *a priori* assumptions.

8.2 Future Work

A number of additional concepts are recommended for future work. These recommendations are of two types: improvements to the trajectory model and improvements to the performance of the solvers. The HOCP mission design automaton described in this work is already an example of how multiple-flyby interplanetary trajectories may be generated automatically with no user interaction, but the following recommendations would allow more types of trajectories to be modeled and might possibly improve the efficiency of the inner-loop problem solvers.

8.2.1 Model Improvements

This work presents HOCP mission design automata using three trajectory models: MGA, MGA-DSM, and MGA-LT. These three models may be used for missions that allow an impulsive-thrust maneuver at the periapse of each flyby or at any point between each pair of planets, or that use continuous-thrust propulsion, respectively. However none of these models may be used to design a mission with two or more impulsive-thrust maneuvers in a given phase. For example, consider the case of a spacecraft entering the sphere of influence of Saturn on its way to rendezvous with Titan. One possible mission design would include an impulsive-thrust maneuver at the closest approach to Saturn that would alter the spacecraft's orbit from a hyperbola to a highly eccentric ellipse, and then a second impulsive-thrust maneuver at apoapse to target a Titan flyby. The Titan flyby could be used to alter the spacecraft's trajectory so that it now orbits in the same plane as Saturn, and then one last impulsive-thrust maneuver could be used to target the Titan rendezvous. This mission cannot be

represented in any of the current trajectory models because the first phase, from the boundary of Saturn's sphere of influence to the Titan flyby, contains two impulsive-thrust maneuvers. It is therefore recommended that a trajectory model be developed that can accommodate any number of impulsive-thrust maneuvers, and that the outer-loop be modified to choose the number of impulses as well as the flyby sequence. This could be done either by adding a second set of outer-loop decision variables to control the number of impulses or by adding additional integer codes to the existing flyby "menu" so that, for example, there might be several codes representing Titan. One might represent "Titan flyby with one maneuver," one might represent "Titan flyby with two maneuvers," *etc.*. Either or both of these methods might be useful in increasing the flexibility of the existing automaton.

Also, the current HOCP mission design automaton can currently be used to design missions only in the reference frame of the Sun. This is adequate for research and development purposes, but future researchers may wish to design a mission to travel, for example, among the moons of Jupiter. Furthermore there may be a need for a tool that can model missions that begin in a parking orbit about the Earth, travel to another planet, and then insert into an orbit about that planet or even about moons of a large planet like Jupiter. Such a mission would require modeling of three or even four reference frames (Earth, Sun, Jupiter, Jovian moon) and the automaton would need to be able to switch between reference frames as needed.

8.2.2 Inner-Loop Solver Improvements

There are four principal areas that should be investigated in order to improve the performance of the inner-loop problem solver. First, MBH has been demonstrated to be so successful for solving MGA-LT problems with many decision variables and nonlinear constraints that it may be worthwhile to adopt MBH as a solver for MGA and MGA-DSM problems, which have comparatively few decision variables and con-

straints, as well. In the existing HOCP automaton, this a very non-trivial change. However if a new automaton were to be developed, MBH should be investigated as a solver for all types of problems.

Second, alternative probability distributions may improve the performance of MBH. A preliminary investigation showed that drawing the "hop" distances from a Cauchy distribution rather than a uniform distribution seemed to produce good results, but a formal, exhaustive benchmark was never performed because it would have taken months of run time. However, if such a study were performed by a future researcher, it may be possible to improve the efficiency of MBH. There may exist probability distributions that are even better than the Cauchy distribution.

Third, it may be possible to improve the robustness of the MGA-LT inner-loop solver by providing analytical expressions for the partial derivatives of the objective function and constraints with respect to the decision variables. Nonlinear programming solvers such as SNOPT require derivative information. Currently SNOPT calculates the derivatives for the MGA-LT problem via finite differencing. However, the SNOPT users manual [70] suggests that if the values of the derivatives are supplied instead of being calculated via finite differencing, SNOPT will perform better. It is very challenging to define these derivatives, especially those of the match point continuity constraints, but perhaps an automated method could be constructed to assemble the derivatives for any MGA-LT problem much in the same way as upper and lower bounds on the decision variables are currently created.

Fourth, it would be worthwhile to further investigate running the automaton in parallel. Currently the MGA and MGA-DSM versions of the EMTG work well in parallel but the MGA-LT version does not. This is because SNOPT, the NLP problem solver used with MBH to solve the inner-loop problem, does not work in a shared memory parallel framework. This is a common problem, and in fact other important components such as high fidelity ephemeris software do not work in a shared mem-

ory environment. One possible solution to this problem is to move to a distributed memory parallel environment such as a computing cluster.

References

[1] Williams, D., "Chronology of Lunar and Planetary Exploration," January 2012, http://nssdc.gsfc.nasa.gov/planetary/chronology.html.

[2] "Jupiter Icy Moons Orbiter," June 2012, http://www.jpl.nasa.gov/jimo/.

[3] Oleson, S. and McGuire, M., "COMPASS Final Report: Saturn Moons Orbiter Using Radioisotope Electric Propulsion (REP): Flagship Class Mission," Tech. rep., NASA Glenn Research Center, 2011.

[4] "Deep Space 1," May 2012, http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1998-061A.

[5] "Hayabusa," November 2012, http://www.isas.jaxa.jp/e/enterp/missions/hayabusa/index.shtml.

[6] Russell, C., Barucci, M., Binzel, R., Capria, M., Christensen, U., Coradini, A., De Sanctis, M., Feldman, W., Jaumann, R., Keller, H., Konopliv, A., McCord, T., McFadden, L., McKeegan, K., McSween, H., Mottola, S., Nathues, A., Neukum, G., Pieters, C., Prettyman, T., Raymond, C., Sierks, H., Smith, D., Spohn, T., Sykes, M., Vilas, F., and Zuber, M., "Exploring the asteroid belt with ion propulsion: Dawn mission history, status and plans," *Advances in Space Research*, Vol. 40, No. 2, 2007, pp. 193 – 201.

[7] Minovitch, M., "The Determination and Characteristics of Ballistic Interplanetary Trajectories Under the Influence of Multiple Planetary Attractions," Tech. rep., Jet Propulsion Lab Technical Report 32-464, 1963.

[8] D'Amario, L. A., Bright, L. E., and Wolf, A. A., "Galileo trajectory design," *Space Science Reviews*, Vol. 60, No. 1-4, 1992, pp. 23–78.

[9] "Galileo Trajectory," June 2012, http://www.jpl.nasa.gov/jplhistory/captions/galileotrajectory-t.php.

[10] Peralta, F. and Flanagan, S., "Cassini interplanetary trajectory design," *Control Engineering Practice*, Vol. 3, No. 11, 1995, pp. 1603–1610.

[11] Paul, M. V. and Finnegan, E. J., "A highly constrained mission to the innermost Planet," *IEEE Aerospace Conference Proceedings*, 2008.

[12] "Rosetta," June 2012, http://www.esa.int/esaMI/Rosetta/.

[13] Guo, Y. and Farquhar, R. W., "New Horizons Mission Design," *Space Science Reviews*, Vol. 140, No. 1-4, OCT 2008, pp. 49–74.

[14] Vasile, M. and De Pascale, P., "Preliminary Design of Multiple Gravity-Assist Trajectories," *Journal of Spacecraft and Rockets*, Vol. 43, No. 4, 2006, pp. 794–805.

[15] Vinkó, T. and Izzo, D., "Global Optimisation Heuristics and Test Problems for Preliminary Spacecraft Trajectory Design," Tech. Rep. GOHTPPSTD, European Space Agency, the Advanced Concepts Team, 2008, Available on line at www.esa.int/act.

[16] Schlueter, M., Rckmann, J., and Gerdts, M., "Non-linear mixed-integer-based Optimisation Technique for Space Applications," Poster for ESA NPI Day, 2010.

[17] Vasile, M., Minisci, E., and Locatelli, M., "Analysis of some global optimization algorithms for space trajectory design," *Journal of Spacecraft and Rockets*, Vol. 47, No. 2, 2010, pp. 334–344.

[18] Ceriotti, M. and Vasile, M., "Automated multigravity assist trajectory planning with a modified ant colony algorithm," *Journal of Aerospace Computing, Information and Communication*, Vol. 7, No. 9, 2010, pp. 261–293.

[19] Olds, A. D., Kluever, C. A., and Cupples, M. L., "Interplanetary mission design using differential evolution," *Journal of Spacecraft and Rockets*, Vol. 44, No. 5, 2007, pp. 1060–1070.

[20] Bryson, A. and Ho, Y., *Applied Optimal Control*, Taylor and Francis, 1975.

[21] Enright, P. J. and Conway, B. A., "Optimal finite-thrust spacecraft trajectories using collocation and nonlinear programming," *Journal of Guidance, Control, and Dynamics*, Vol. 14, No. 5, 1991, pp. 981 – 985.

[22] Enright, P. and Conway, B., "Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 4, July-August 1992, pp. 994–1002.

[23] Vasile, M. and Bernelli-Zazzera, F., "Optimizing low-thrust and gravity assist maneuvers to design interplanetary trajectories," *Journal of the Astronautical Sciences*, Vol. 51, No. 1, January-March 2003, pp. 13–35.

[24] Sims, J. A. and Flanagan, S. N., "Preliminary Design of Low-Thrust Interplanetary Missions," *AAS/AIAA Astrodynamics Specialist Conference*, Girdwood, Alaska, August 1999.

[25] Vasile, M. and Campagnola, S., "Design of low-thrust multi-gravity assist trajectories to Europa," *Journal of the British Interplanetary Society*, Vol. 62, No. 1, 2009, pp. 15–31.

[26] Petropoulos, A. and Longuski, J., "Shape-based algorithm for automated design of low-thrust, gravity-assist trajectories," *Journal of Spacecraft and Rockets*, Vol. 41, No. 5, September-October 2004, pp. 787–796.

[27] Wall, B. and Conway, B., "Shape-Based Approach to Low-Thrust Rendezvous Trajectory Design," *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 1, 2009, pp. 95–101.

[28] Novak, D. and Vasile, M., "Improved Shaping Approach to the Preliminary Design of Low-Thrust Trajectories," *Journal of Guidance, Control, and Dynamics*, Vol. 34, No. 1, 2010, pp. 128–147.

[29] Conway, B., Chilan, C., and Wall, B., "Evolutionary principles applied to mission planning problems," *Celestial Mechanics and Dynamical Astronomy*, Vol. 97, No. 2, 2007, pp. 73 – 86.

[30] Ghosh, P. and Conway, B. A., "A Direct Method for Trajectory Optimization Using the Particle Swarm Approach," *21 st. AAS/AIAA Space Flight Mechanics Meeting*, New Orleans, Louisiana, February 2011.

[31] Coverstone-Carroll, V., Hartmann, J., and Mason, W., "Optimal multi-objective low-thrust spacecraft trajectories," *Computer Methods in Applied Mechanics and Engineering*, Vol. 186, No. 24, 2000, pp. 387 – 402.

[32] Woo, B., Coverstone, V., and Cupples, M., "Low-thrust trajectory optimization procedure for gravity-assist, outer-planet missions," *Journal of Spacecraft and Rockets*, Vol. 43, 2006, pp. 121–129.

[33] Vavrina, M. and Howell, K., "Global Low Thrust Trajectory Optimization through Hybridization of a Genetic Algorithm and a Direct Method," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Honolulu, Hawaii, August 18-21 2008.

[34] Yam, C., di Lorenzo, D., and Izzo, D., "Low-Thrust Trajectory Design as a Constrained Global Optimization Problem," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 225, 2011, pp. 1243–1251.

[35] Longuski, J. and Williams, S., "Automated design of gravity-assist trajectories to Mars and the outer planets," *Celestial Mechanics and Dynamical Astronomy*, Vol. 52, No. 3, 1991, pp. 207–220.

[36] Ceriotti, M. and Vasile, M., "MGA trajectory planning with an ACO-inspired algorithm," *Acta Astronautica*, Vol. 67, No. 910, 2010, pp. 1202 – 1217.

[37] Gad, A. and Abdelkhalik, O., "Hidden Genes Genetic Algorithm for Multi-Gravity-Assist Trajectories Optimization," *Journal of Spacecraft and Rockets*, Vol. 48, No. 4, July-August 2011, pp. 629–641.

[38] Abdelkhalik, O. and Gad, A., "Dynamic-Size Multi-Population Genetic Optimization for Multi-Gravity-Assist Trajectories," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 2, March-April 2012, pp. 520529.

[39] Alemany, K. and Braun, R., "Design space pruning heuristics and global optimization method for conceptual design of low-thrust asteroid tour missions," *International Astronautical Federation - 59th International Astronautical Congress*, 2008.

[40] von Stryk, O. and Glocker, M., "Numerical Mixed-Integer Optimal Control and Motorized Traveling Salesmen Problems," *APII-JESA (Journal europen des systmes automatiss - European Journal of Control)*, Vol. 35, No. 4, 2001, pp. 519–533.

[41] Chilan, C., *Automated Design of Multiphase Spacecraft Missions Using Hybrid Optimal Control*, PhD dissertation, Department of Aerospace Engineering, University of Illinois, 2009.

[42] Ross, I. M. and D'Souza, C., "Hybrid Optimal Control Framework for Mission Planning," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 4, July-August 2005, pp. 686–697.

[43] Buss, M., Glocker, M., Hardt, M., von Stryk, O., Bulirsch, R., and Schmidt, G., "Nonlinear Hybrid Dynamical Systems: Modeling, Optimal Control, and Applications," *Lecture Notes in Control and Information Sciences*, Vol. 279, 2002, pp. 311–335.

[44] Wall, B., *Technology for the Solution of Hybrid Optimal Control Problems in Astronautics*, PhD dissertation, Department of Aerospace Engineering, University of Illinois, 2007.

[45] Chilan, C. M. and Conway, B. A., "A space mission automaton using hybrid optimal control," *Advances in the Astronautical Sciences*, Vol. 127 PART 1, 2007, pp. 259–276.

[46] Downing, J., "Pyxis Tool," Tech. Rep. NASA/TM-2006-214139, Flight Dynamics Branch (FDAB), NASA Goddard Space Flight Center, August 2006.

[47] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[48] Levine, D., "Users Guide for PGAPack," January 1996, `nfo.mcs.anl.gov/pub/tech_reports/reports/ANL9518.ps.Z`.

[49] "GALib," October 2012, `http://lancet.mit.edu/ga/`.

[50] Igel, C., Glasmachers, T., and Heidrich-Meisner, V., "Shark," *Journal of Machine Learning Research*, Vol. 9, 2008, pp. 993–996.

[51] "MATLAB," October 2012, `http://www.mathworks.com/products/matlab/`.

[52] "Global Optimization Toolbox," October 2012, `http://www.mathworks.com/products/global-optimization/`.

[53] Kajitani, I., Hoshino, T., Iwata, M., and Higuchi, T., "Variable length chromosome GA for evolvable hardware," *Proceedings of the IEEE International Conference on Evolutionary Computation*, May 20-22 1996.

[54] Chopra, V. S., "Chromosomal organization at the level of gene complexes," *Cellular and Molecular Life Sciences*, 2010, pp. 1–14.

[55] Prussing, J. and Conway, B., *Orbital Mechanics*, Oxford University Press, New York, 1993.

[56] "Global Trajectory Optimization Problem Database," October 2012, `http://www.esa.int/gsp/ACT/inf/op/globopt.htm`.

[57] Kennedy, J. and Eberhart, R., "Particle swarm optimization," *IEEE International Conference on Neural Networks - Conference Proceedings*, Vol. 4, 1995, pp. 1942–1948.

[58] Storn, R. and Price, K., "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, Vol. 11, No. 4, 1997, pp. 341–359.

[59] Price, K. and Storn, R., "Differential Evolution (DE) for Continuous Function Optimization," October 2012, `http://www1.icsi.berkeley.edu/~storn/code.html`.

[60] Oleson, S. and McGuire, M., "COMPASS Final Report: Enceladus Solar Electric Propulsion Stage," Tech. rep., NASA Glenn Research Center, 2011.

[61] Hubbard, W. B., "Ice Giants Decadal Study," Tech. rep., NASA Planetary Sciences Decadal Survey, June 2010, `http://sites.nationalacademies.org/SSB/SSB_059331`.

[62] Wall, B. and Novak, D., "A 3D Shape-Based Approximation Method for Low-Thrust Trajectory Design," *AAS Astrodynamics Specialist Conference, Girdwood, AK*, July 2011.

[63] Wall, B., Pols, B., and Lanktree, B., "Shape-Based Approximation Method for Low-Thrust Interception and Rendezvous Trajectory Design," *AAS/AIAA Space Flight Mechanics Meeting*, San Diego, CA, February 2010.

[64] Sims, J., Finlayson, P., Rinderle, E., Vavrina, M., and Kowalkowski, T., "Implementation of a low-thrust trajectory optimization algorithm for preliminary design," *AIAA/AAS Astrodynamics Specialist Conference*, August 2006.

[65] "PaGMO (Parallel Global Multiobjective Optimizer)," March 2012, `http://pagmo.sourceforge.net/pagmo/index.html`.

[66] "NASA Launch Services Program Performance Web Site," March 2012, `http://elvperf.ksc.nasa.gov/elvMap/staticPages/launch_vehicle_info1.html`.

[67] Hofer, R., "High-Specific Impulse Operation of the BPT-4000 Hall Thruster for NASA Science Missions," *46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Nashville, TN*, July 2010.

[68] Oh, D. and Goebel, D., "Performance evaluation of an expended range XIPS ion thruster system for NASA science missions," *42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Sacramento, CA*, July 2006.

[69] Oh, D., Benson, S., Witzberger, K., and Cupples, M., "Performance evaluation of an expended range XIPS ion thruster system for NASA science missions," *40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Ft. Lauderdale, FL*, July 2004.

[70] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Rev.*, Vol. 47, No. 1, 2005, pp. 99–131.

[71] Enright, P. and Conway, B., "Discrete Approximations to Optimal Trajectories Using Collocation and Nonlinear Programming," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 4, 1992, pp. 994 – 1002.

[72] Leary, R., "Global optimization on funneling landscapes," *Journal of Global Optimization*, Vol. 18, No. 4, December 2000, pp. 367–383.

[73] Vasile, M., Minisci, E., and Locatelli, M., "Analysis of Some Global Optimization Algorithms for Space Trajectory Design," *Journal of Spacecraft and Rockets*, Vol. 47, No. 2, March-April 2010, pp. 334–344.

[74] Addis, B., Cassioli, A., Locatelli, M., and Schoen, F., "A global optimization method for the design of space trajectories," *Computational Optimization and Applications*, Vol. 48, No. 3, April 2011, pp. 635–652.

[75] Viswanathan, G. M., da Luz, M., Rapaso, E., and Stanley, H., *The Physics of Foraging*, Cambridge University Press, 2011.

[76] Greeley, R. and Johnson, T., "Report of the NASA Science Definition Team for the Jupiter Icy Moons Orbiter," Tech. rep., Jupiter Icy Moons Orbiter Science Definition Team, February 2004, `http://ossim.hq.nasa.gov/jimo/JIMO_SDT_REPORT.pdf`.

[77] Whiffen, G. J., "An Investigation of a Jupiter Galilean Moon Orbiter Trajectory," *AAS/AIAA Astrodynamics Specialist Conference*, August 2003.

[78] Yam, C. H., McConaghy, T., Chen, K. J., and Longuski, J., "Preliminary Design of Nuclear Electric Propulsion Missions to the Outer Planets," *AIAA/AAS Astrodynamics Specialist Conference*, August 2004.

[79] Yam, C. H., McConaghy, T., Chen, K. J., and Longuski, J., "Design of Low-Thrust Gravity-Assist Trajectories to the Outer Planets," *55th International Astronautical Congress*, October 2004.

[80] Parcher, D. and Sims, J., "Gravity-Assist Trajectories to Jupiter Using Nuclear Electric Propulsion," *AAS/AIAA Astrodynamics Specialist Conference*, August 2005.

[81] Garcia Yarnoz, D., Jehn, R., and Croon, M., "Interplanetary Navigation Along the Low-Thrust Trajectory of BepiColombo," *Acta Astronautica*, Vol. 59, 2006, pp. 284–293.

[82] Katzkowski, M., Corral, C., Jehn, R., Pellon, J.-L., Khan, M., Yanez, A., and Biesbroek, R., "BepiColombo Mercury Cornerstone Mission Analysis: Input to Definition Study," Tech. rep., April 2002.