GLOBAL OPTIMIZATION OF MGA-DSM PROBLEMS USING THE INTERPLANETARY GRAVITY ASSIST TRAJECTORY OPTIMIZER (IGATO)

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Aerospace Engineering

by
Jason M. Bryan
December 2011
### COMMITTEE MEMBERSHIP

**TITLE:** Global optimization of MGA-DSM problems using the Interplanetary Gravity Assist Trajectory Optimizer (IGATO)

**AUTHOR:** Jason M. Bryan

**DATE SUBMITTED:** December 2011

**COMMITTEE CHAIR:** Dr. Kira Abercromby

**COMMITTEE MEMBER:** Dr. David Marshall

**COMMITTEE MEMBER:** Dr. John Keller

**COMMITTEE MEMBER:** Dr. Eric Mehiel
ABSTRACT

Global optimization of MGA-DSM problems using the Interplanetary Gravity Assist Trajectory Optimizer (IGATO)

Jason M. Bryan

Interplanetary multiple gravity assist (MGA) trajectory optimization has long been a field of interest to space scientists and engineers. Gravity assist maneuvers alter a spacecraft's velocity vector and potentially allow spacecraft to achieve changes in velocity ($\Delta V$) which would otherwise be unfeasible given our current technological limitations. Unfortunately, designing MGA trajectories is difficult and in order to find good solutions, deep space maneuvers (DSM) are often required which further increase the complexity of the problem. In addition, despite the active research in the field over the last 50 years, software for MGA trajectory optimization is scarce. A few good commercial, and even fewer open-source, options exist, but a majority of quality software remains proprietary.

The intent of this thesis is twofold. The first part of this work explores the realm of global optimization applied to multiple gravity assist trajectories with deep space maneuvers (MGA-DSM). With the constant influx of new global optimization algorithms and heuristics being developed in the global optimization community, this work aims to be a high level optimization approach which makes use of those algorithms instead of trying to be one itself. Central to this approach is PaGMO, which is the open-source Parallel Multiobjective Global Optimizer created by ESA's Advanced Concepts Team (ACT). PaGMO is an implementation of the Island Model Paradigm which allows the parallelization of different global optimizers. The second part of this work introduces the IGATO software which improves PaGMO by complementing it with dynamic restart capabilities, a pruning algorithm which learns over time, subdomain decomposition, and other techniques to create a powerful optimization tool. IGATO aims to be an open-source platform independent C++ application with a robust graphical user interface (GUI). The application is equipped with 2D plotting and simulations, real time Porkchop Plot generation, and other useful features for analyzing various problems. The optimizer is tested on several challenging MGA-DSM problems and performs well: consistently performing as well or better than PaGMO on its own.

Keywords: Gravity assist, interplanetary trajectory, global optimization, PaGMO, MGA
ACKNOWLEDGMENTS

First and foremost I would like to acknowledge my advisor Dr. Kira Abercromby for her continuous support and patience. Global optimization and interplanetary gravity assist trajectory design are both tremendously broad subjects and this thesis changed course on multiple occasions as lessons were learned and new discoveries were made. My advisor was always supportive of the choices I made and the new directions I took and I thank her for it.

I would also like to thank the European Space Agency's Advanced Concepts Team for creating the GTOP Database which is a huge contribution to the MGA-DSM optimization community, along with all of the developers of PaGMO for creating such a fantastic open-source global optimization tool. Your hard work and dedication are what made this thesis possible.

Lastly, I would like to thank SpaceWorks Software and SpaceFlightSolutions for allowing me to use their great software for research.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES ............................................................................... x</td>
</tr>
<tr>
<td>LIST OF FIGURES ........................................................................... xii</td>
</tr>
</tbody>
</table>

## CHAPTER

### I. INTRODUCTION ................................................................. 1

| I.I Statement of Purpose ........................................................... 2 |
| I.II Existing Software ............................................................... 4 |
| I.III Organization of the Report .................................................. 6 |

### II. TRAJECTORY MODEL ....................................................... 7

| II.I Introduction ............................................................................... 7 |
| II.II Conic Arcs ............................................................................... 8 |
| II.II.I Lambert's Arc ....................................................................... 8 |
| II.II.II Propagation Arc .................................................................. 10 |
| II.III ∆V Maneuvers ..................................................................... 11 |
| II.III.I Launch ............................................................................... 11 |
| II.III.II Rendezvous ..................................................................... 13 |
| II.III.III Orbital Departure ........................................................... 14 |
| II.III.IV Orbital Insertion .............................................................. 15 |
| II.III.V Flyby ................................................................................. 16 |
| II.III.VI Deep Space Maneuver (DSM) ........................................... 19 |

### II.IV Velocity Formulation ...................................................... 19

| II.IV.I Trajectory Parameterization .............................................. 22 |

### III. GLOBAL OPTIMIZATION ................................................. 29

| III.I Introduction ........................................................................... 29 |
| III.II The Search Space ............................................................... 31 |
III.III Algorithms .......................................................... 36
III.IV The Island Model & PaGMO ........................................ 46
III.V GTOP Database .......................................................... 48

IV. IGATO OPTIMIZATION ENGINE ........................................ 49
   IV.I Introduction .............................................................. 49
   IV.II Restarts ................................................................. 49
   IV.III Interval Based Pruning ............................................. 52
       IV.III.I Interval Archiving ........................................... 54
       IV.III.II The Affine Space ............................................ 57
   IV.IV Subdomain Decomposition ....................................... 61
   IV.V Similarity Testing ................................................... 64
   IV.VI Optimization Routine Summary ................................. 70

V. TEST CASES & VALIDATION ............................................. 76
   V.I Setup ........................................................................ 76
   V.II GTOP Test Cases ..................................................... 80
       V.II.I Case 1: Messenger ............................................... 81
       V.II.II Case 2: Cassini 2 ............................................... 89
       V.II.III Case 3: Messenger Full .................................... 94
   V.III Additional Test Cases .............................................. 99
       V.III.I Case 4: Mars Round-Trip ................................... 99

VI. SOFTWARE ................................................................. 102
   VI.I Design Choices ...................................................... 102
   VI.II IGATO GUI User Guide ........................................... 104
       VI.II.I Trajectory Inputs .............................................. 104
       VI.II.II Optimizer Settings .......................................... 111
       VI.II.III Output ......................................................... 115
       VI.II.IV Real Time Archipelago Analysis ....................... 121
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Comparison between the intervals and node points for an increasing number of dimensions</td>
<td>54</td>
</tr>
<tr>
<td>4.2</td>
<td>The first six solutions to the MATLAB Peaks function an arbitrary global optimizer discovered</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Parameter configuration of the Adaptive Simulated Annealing algorithm (SA Corana) in PaGMO</td>
<td>78</td>
</tr>
<tr>
<td>5.2</td>
<td>Parameter configuration of the Differential Evolution algorithm (DE variant 2) in PaGMO</td>
<td>79</td>
</tr>
<tr>
<td>5.3</td>
<td>Parameter configuration of the Particle Swarm Optimization algorithm (PSO variant 5) in PaGMO</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>Parameter configuration of the Differential Evolution algorithm (DE variant 5) in PaGMO</td>
<td>79</td>
</tr>
<tr>
<td>5.5</td>
<td>The state vector bounds for the Messenger problem from the GTOP database .</td>
<td>82</td>
</tr>
<tr>
<td>5.6</td>
<td>The three best known solutions to the Messenger problem from the GTOP database along with their corresponding state vectors</td>
<td>83</td>
</tr>
<tr>
<td>5.7</td>
<td>The IGATO optimization settings for the Messenger and Cassini2 test cases .</td>
<td>85</td>
</tr>
<tr>
<td>5.8</td>
<td>PaGMO results for GTOP Messenger problem</td>
<td>86</td>
</tr>
<tr>
<td>5.9</td>
<td>IGATO results for the GTOP Messenger problem</td>
<td>87</td>
</tr>
<tr>
<td>5.10</td>
<td>Comparison between IGATO and PaGMO's success rates on the GTOP Messenger problem</td>
<td>87</td>
</tr>
<tr>
<td>5.11</td>
<td>The state vector bounds for the Cassini2 problem from the GTOP database .</td>
<td>89</td>
</tr>
<tr>
<td>5.12</td>
<td>The best known solution to the Cassini2 problem from the GTOP database .</td>
<td>90</td>
</tr>
<tr>
<td>5.13</td>
<td>PaGMO results for GTOP Cassini2 problem</td>
<td>91</td>
</tr>
<tr>
<td>5.14</td>
<td>IGATO results for the GTOP Cassini2 problem</td>
<td>92</td>
</tr>
<tr>
<td>5.15</td>
<td>Comparison between IGATO and PaGMO's success rates on the GTOP Cassini2 problem</td>
<td>92</td>
</tr>
<tr>
<td>5.16</td>
<td>State vector bounds on the Messenger Full problem from the GTOP database .</td>
<td>94</td>
</tr>
<tr>
<td>5.17</td>
<td>IGATO optimization settings for the Messenger Full test case .</td>
<td>96</td>
</tr>
<tr>
<td>5.18</td>
<td>PaGMO results for GTOP Messenger Full problem</td>
<td>97</td>
</tr>
</tbody>
</table>
5.19. IGATO results for the GTOP Messenger Full problem ........................................ 97
5.20. Comparison between IGATO and PaGMO's success rates on the GTOP Messenger Full problem .................................................................................. 97
5.21. The state vector bounds for the Mars Round-Trip problem ................................ 100
5.22. The IGATO optimization settings for the Mars Round-Trip problem ............ 101
5.23. IGATO results for the Mars Round-Trip problem ............................................. 101
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Lambert's Problem: determining the conic arc between two position vectors</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>in a finite amount of time</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>Geometry of the flyby</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Example interplanetary trajectory consisting of a launch, DSM, unpowered</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>flyby, and rendezvous</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Visualization of the search space and fitness landscape</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Rastrigin's function demonstrating a highly multimodal fitness landscape</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Rastrigin's function with added noise. Left: n=5. Right: n=20</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>The search space in two dimensions with four intervals along each dimension.</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>The dotted lines represent the grid which is created</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Demonstration of the interval archiving procedure for the first three solutions from Table 4.2</td>
<td>56</td>
</tr>
<tr>
<td>4.3</td>
<td>Demonstration of the interval archiving procedure for the last three solutions from Table 4.2</td>
<td>57</td>
</tr>
<tr>
<td>4.4</td>
<td>The affine transformation applied to two discontinuous intervals of different length (left). The resulting intervals (right) are continuous and equally spaced.</td>
<td>59</td>
</tr>
<tr>
<td>4.5</td>
<td>The wrapped objective function. Each decision vector must be transformed</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>back into the real space before the objective function can be calculated</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>A coordinate in the affine space with four intervals. Their boundaries are</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>always the same</td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>An active subdomain of entire domain. Performing an evolution within the</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>subdomain restricts the optimizer from finding solutions outside the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subdomain boundaries</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>An overview of the optimization routine</td>
<td>73</td>
</tr>
<tr>
<td>4.9</td>
<td>An overview of the Optimizer class and its children in IGATO along with</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>some of their major methods</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>The 'Rim' island topology used for all test cases for IGATO and PaGMO</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>along with the algorithm and migration policy assigned to each island</td>
<td></td>
</tr>
</tbody>
</table>
5.2. The levels of the subdomain decomposition algorithm. Each box represents a subdomain and after each level, each subdomain splits into two children subdomains .......................................................................................................... 81

5.3. IGATO trajectory plot of the best found solution (f=8.630 km/s) to the GTOP Messenger problem ........................................................................................................................................... 88

5.4. IGATO Porkchop plot for the GTOP Messenger problem ........................................ 88

5.5. IGATO trajectory plot for the best found solution (f=8.3830 km/s) to the Cassini2 GTOP problem ........................................................................................................................................ 93

5.6. IGATO Porkchop plot for the Cassini2 GTOP problem ........................................ 93

5.7. IGATO trajectory plot for the best found trajectory (f=7.0584 km/s) for the Messenger Full GTOP problem .............................................................................................................................................. 98

5.8. IGATO Porkchop plot of the Messenger Full GTOP problem .............................. 98

5.9. IGATO trajectory plot for the best found solution (f=6.3065 km/s) for the Mars Round-Trip problem .................................................................................................................................................... 102

5.10. IGATO Porkchop plot for the Mars Round-Trip problem ................................. 102

6.1. The main window with the Trajectory Inputs tab ................................................. 106

6.2. The Trajectory Inputs tab with a departure object selected ................................ 108

6.3. The Trajectory Inputs tab with an arrival object selected .................................. 110

6.4. The Trajectory Inputs tab with a flyby object selected ..................................... 112

6.5. The Optimizer Inputs tab .................................................................................. 116

6.6. The Simulation Output sub-tab of the Output tab .............................................. 120

6.7. The Solution Summary sub-tab of the Output tab ............................................. 121

6.8. The Trajectory Details sub-tab of the Output tab .............................................. 121

6.9. The Trajectory Plot sub-tab of the Output tab .................................................. 122

6.10. The Population Breakdown tab ........................................................................ 126

6.11. The Fitness Trends tab ..................................................................................... 128

6.12. The Solution Summary tab ............................................................................... 129
6.13. The Best Solution Dialog pop-up window ......................................................... 130
6.14. The Population Breakdown tab with a best known solution inserted (red) ...... 130
6.15. IGATO generated Porkchop plot of the Cassini2 problem from the GTOP
database .............................................................................................................. 132
6.16. The Output file generated by IGATO .............................................................. 133
I. Introduction

Given the past and current limitations on a spacecraft’s propulsion capabilities, scientists and engineers have long studied creative ways to push a spacecraft past its technological boundaries. Possibly the greatest breakthrough was the idea of a gravity assist maneuver (also called a flyby or gravitational slingshot). Gravity assist maneuvers involve exchanging energy or momentum with a celestial body as a spacecraft flies past it. As long as an approaching spacecraft does not crash into the celestial body or get captured into a parking orbit, it will continue past the planet on a hyperbolic trajectory [22]. The idea is that, during this hyperbolic trajectory, there will occur a momentum exchange between the spacecraft and the celestial body. Since the mass of the celestial body is typically quite large in contrast to the spacecraft, the small subtraction or addition of momentum results in a negligible change its net velocity. However, since the mass of the spacecraft is extremely small compared to that of the celestial body, the velocity imparted on the spacecraft from the celestial body can be significant. Whether the spacecraft gains momentum or loses momentum entirely depends on the direction that it flies past the celestial body. If the spacecraft crosses in front of the celestial body's direction of motion, the flyby is referred to as a leading-side flyby and the spacecraft will lose heliocentric velocity. On the contrary, if the spacecraft crosses behind the celestial body's direction of motion, it's called a trailing-side flyby and the spacecraft gains heliocentric velocity [22]. The direction of a spacecraft's relative velocity vector as it approaches a celestial body also determines how much it "gains" from the flyby. If the spacecraft's relative velocity
vector is nearly parallel to the celestial body's velocity vector, then the gain will be small, whereas if it is nearly perpendicular than it may be quite large. Thus, gravity assist maneuvers have the potential to drastically change a spacecraft orbital energy simply depending on how it approaches a celestial body. This essentially allows a spacecraft to reach destinations that might otherwise be unfeasible given our technological limitations, at virtually no extra cost. Well actually, there is a cost to gravity assist maneuvers: They are very difficult nonlinear problems with no closed-form analytic solution. The difficulty with gravity assist maneuvers boils down to their dependency on desirable alignments of celestial bodies (namely planets) which, depending on their synodic period, are often hard to come by. The problem escalates tremendously when multiple gravity assist maneuvers are required for a mission. Over the last 60 years there has been a great deal of interest and research in gravity assist maneuvers and many flown spacecraft have successfully taken advantage of them. In the past, planning multiple gravity assist (MGA) trajectories took considerable experience and expertise with celestial mechanics, often requiring a combination of studying charts and crunching numbers with supercomputers. Fortunately, as computers have grown considerably more powerful in the last 20 years, new methods and tools have been developed which makes optimizing one or even multiple gravity assist trajectories much easier and possible even on a personal computer. For more information on the history of spacecraft trajectory optimization refer to [1].

I.I. Statement of Purpose

One of the drawbacks of spacecraft interplanetary trajectory optimization is that its usage is still extremely limited. The vast majority of the aerospace industry is primarily
concerned with Earth orbiting spacecraft which does not provoke the need for interplanetary trajectory optimization software. The interplanetary missions that have been conducted are typically government projects and much of the software that has been developed and used for these missions is proprietary. A few commercial options are available and very little open-source software is available at all (see Section I.II). Therefore, the intent of this thesis was to explore the field of interplanetary multiple gravity assist trajectory optimization using global optimization and develop a cross-platform, open-source, and user-friendly application. This resulting application is called IGATO (Interplanetary Gravity Assist Trajectory Optimizer). In order to not reinvent the wheel, this software builds upon the open-source parallel optimization framework PaGMO (see Section III.IV) and concentrates on improving its performance when applied to interplanetary trajectory optimization.

The software has a number of design objectives:

- The software should be standalone, cross-platform, and open-source.
- The software should be equipped with a robust graphical user interface (GUI) to create a user-friendly interface for constructing interplanetary missions and which is capable of handling its own trajectory plotting and simulations.
- The software should be capable of consistently identifying the best known solution (or close to it) to common multiple gravity assist with deep space maneuvers (MGA-DSM) problems.
- The software should be able to identify a variety of quality alternate solutions for different launch windows.
- The software should make use of modern computer's parallelization capabilities.
A user manual should be provided for new user's who wish to make use of the software.

I.II. Existing Software

There is an obvious lack of commercial-off-the-shelf (CoTS) or open-source (free) interplanetary trajectory optimization software currently available to the public. Only three commercially available solvers and open-source solver were found (although more most certainly exist). These applications are described below. For a more complete list of global optimization solvers that have been applied to spacecraft trajectory optimization in the past refer to [1].

Bullseye

Bullseye is a commercial application developed by SpaceWorks Software [25]. Bullseye offers simple and clean graphical user interface for optimizing interplanetary trajectories. Setting up and running interplanetary scenarios in Bullseye is a breeze and Bullseye features its own 2D trajectory plot and produces a number of useful output files including trajectory details, porkchop plot spreadsheets, and files for exporting solutions to Celestia [28] for 3D simulation. The biggest downside of Bullseye is a one year commercial license will put you back $999.00 with a $299.00 annual renewal fee. A slimmed-down student version is available, but it does not allow multiple flyby trajectories. For more information on Bullseye, refer to [25].
MAnE (Mission Analysis Environment for Heliocentric High-Thrust Missions)

MAnE is fully featured commercial gui application developed by SpaceFlightSolutions [26]. MAnE uses an indirect method based on ordinary calculus to optimize interplanetary trajectories. MAnE also includes a host of utility tools such as a single-rev and multi-rev Lambert solver, Porkchop Plot plotter, and Ephemerides propagator, among many others. MAnE is very versatile when it comes to constructing a trajectory: multiple flybys, DSMs, and surface stay times are all possible and ephemerides is supplied for planets along with a large database of common asteroids and comets. User-defined bodies can also be created and saved. MAnE has many more features and for more information and licensing options refer to [26].

MIDACO (Mixed Integer Distributed Ant Colony Optimization)

MIDACO is a commercial application developed jointly by EADS Astrium and ESA [27]. MIDACO is a black-box optimizer that can be applied to a wide range of global optimization problems. MIDACO uses a variant of Ant Colony Optimization in combination with the Oracle Penalty Method to solve continuous, combinatorial, and mixed integer problems. MIDACO has found several of the best known solutions in the GTOP database (see Section III.V) and is designed for massively parallel architectures making it potentially both fast and effective. MIDACO does not have a graphical user interface, but is available as Fortran, C, or Matlab code. MIDACO has several licensing options ranging from $180-360 for academics to $2400 for commercial use (both single user). For more information on MIDACO refer to [27].
**PaGMO (Parallel Global Multiobjective Optimization)**

PaGMO is an open-source project created by the ESA's Advanced Concepts Team (ACT) [14]. PaGMO is a general optimization framework based on the asynchronous island model paradigm and can be applied to any black-box problem. This work makes extensive use of PaGMO and for more information see Section III.IV.

**I.III. Organization of the Report**

This paper is organized as follows: Chapter two describes the mathematical modeling of interplanetary trajectories, chapter three gives an introduction to the field of global optimization and particularly metaheuristics which are the basis of this work, chapter four walks through the optimization strategy developed in this work, chapter five provides multiple test cases for comparison and validation, and finally chapter six discusses the software implementation and provides a full user's manual.
II. Trajectory Model

This chapter introduces a well-known interplanetary multiple gravity assist with deep space maneuver (MGA-DSM) trajectory model. This chapter also expands on the traditional trajectory model in order to allow for multiple deep space maneuvers between each celestial body along with the support for round-trip surface stay missions. The first section gives an introduction to the trajectory model. The following sections then examine the individual components of the trajectory model in more detail and finally, the last section brings them all together into one unified whole using the method called the velocity formulation.

II.I. Introduction

As with most fields in engineering, the first step in solving a trajectory optimization problem is to model it mathematically. To form a reasonable model, we first need to make a couple key simplifying approximations. First and foremost, we make use of the patched-conic approximation which reduces the problem into a series of connected conic arcs. In order to connect two consecutive arcs together, the end point of the first arc must have the same position and velocity as the starting point of the next arc. If the end point of the first arc and the starting point of the second arc have the same position, but differ in velocity, then we can assume that an instantaneous change in velocity ($\Delta V$) can be used to connect them [1]. The instantaneous change in velocity assumption is reasonably accurate if a maneuver produces a sizable change in velocity in a short amount of time.
This is a good approximation for events such as gravity assist maneuvers, high thrust chemical thrust maneuvers, launch, orbital insertion, etc. Using these two assumptions, the trajectory model simply breaks down into the combination of two types of events: conic arcs and $\Delta V$ maneuvers.

II.II. Conic Arcs

There are two general types of conic arcs that are of interest to us for creating interplanetary trajectories: Lambert's arcs and propagation arcs. Both arcs require us to know the amount of time it takes to travel along the arc from one end point to the other. That being said, we use Lambert's arcs when we only know the position at the endpoints of the arc and we need to find their corresponding velocities. Propagation arcs on the other hand are used when we know the position and velocity at one endpoint and need to determine the state of the other end point.

II.II.I. Lambert's Arcs

A fundamental problem in classical orbital mechanics is determining the conic arc that connects two distinct points $\vec{r}_1, \vec{r}_2$ in space in a finite amount of time $\Delta t$. This problem is formally referred to as Lambert's Problem (see Fig. 2.2):

$$[V_1, V_2] = \text{Lambert}(R_1, R_2, \Delta t, \text{dir})$$

With the conic arc known, the orbital elements are easily deciphered, and the velocities $\vec{v}_1, \vec{v}_2$ can be determined at each end point. In general, there is more than one possible arc that can connect two such points in the desired amount of time, and in certain undesirable situations, there are infinitely many. These undesirable situations occur when the
difference in true anomaly between the two points is a multiple of $180^\circ$. In all other cases, if we limit the direction $dir$ of flight to either prograde or retrograde, then the number of possible arcs is reduced to $2N+1$ where $N$ is the number of full revolutions around the central body. Therefore if no full revolutions are allowed, then there is only one unique solution to Lambert's Problem.

Figure 2.1. Lambert's Problem: determining the conic arc between two position vectors in a finite amount of time.

There is no closed-form solution to Lambert's Problem, and as such, there are numerous variations on the implementation of the algorithm. This work uses the multi-revolution Lambert solver described in [4], but at the time of writing, only single-revolution trajectories are supported in the software.
II.II.II. Propagation Arcs

Another common problem in orbital mechanics is the problem of propagating a spacecraft along an orbit (or from an initial position and velocity) for a desired amount of time.

\[ [R_2, V_2] = Propagate(R_1, V_1, \Delta t) \]

By assuming that there are no perturbations acting on the spacecraft, the problem can easily be solved analytically. Numerical integration can instead be used which can take into account realistic perturbations, but with a loss of performance. For interplanetary trajectories that spend nearly all of their flight time in deep space, preliminary trajectory optimizers can safely disregard these perturbations and still produce reasonably accurate results. There is a third solution which can be used for planets and other celestial bodies and that is to use an ephemerides. An ephemerides is a collection of observed positions for a particular celestial body. Using an ephemerides, the position of a celestial body at a particular time can be estimated by interpolating between past observations or by using extrapolation to predict it at future times. Ephemerides offer the advantage of increased precision over analytic methods, but are typically slower to compute. It is estimated from Ceriotti [1] that using analytic methods are over three times faster to calculate than the widely used NAIF-SPICE ephemerides available from the Jet Propulsion Laboratory's (JPL) website. This work uses analytic methods for fast preliminary design, but the option to use JPL ephemerides in the form of the DE405 or DE421 kernels for higher fidelity trajectories is currently in development. This work assumes knowledge of non-perturbed Keplerian analytical propagation techniques from any classical orbital mechanics textbook [22-24].
II.III. ΔV Maneuvers

A ΔV maneuver is a change in spacecraft's velocity vector. Trajectories are often optimized by minimizing the ΔV because it is analogous to minimizing the required fuel consumption, but without requiring any knowledge of the propulsion system. Transferring from one orbit to another always requires the same amount of ΔV, whereas the required fuel mass could be quite different between a chemical and electric propulsion system. Therefore, ΔV maneuvers provide a general and useful optimization criteria for trajectory design. There are several types of ΔV maneuvers that are used in order to construct an interplanetary trajectory: launch, flyby, DSM, orbital departure, orbital insertion, and rendezvous. Most interplanetary trajectories typically begin with a launch and end with an rendezvous, however an orbital departure or insertion can be used instead if the spacecraft is starting from or ending in a parking orbit respectively. Orbital departures and insertions are also useful for round-trip trajectory scenarios as described below. Flybys and DSM are the joints that connect all the Lambert and propagation arcs together in-between the launch and arrival events. This section will describe each type of maneuver in detail.

II.III.I. Launch

The launch is an important part of the interplanetary trajectory. It is the ΔV maneuver provided by a launch vehicle which launches the spacecraft from the surface of a planet (or any celestial body) into an elliptic orbit. If the ΔV maneuver is large enough, the spacecraft will reach the hyperbolic escape velocity and enter a heliocentric elliptic orbit,
otherwise it will remain in a planetocentric elliptic orbit (parking orbit) and will require an additional $\Delta V$ maneuver later on in order to escape the planet's sphere of influence. In this work, reaching hyperbolic escape velocity from the surface of a planet is considered a *launch event* whereas reaching hyperbolic escape velocity from a parking orbit is considered an *orbital departure* (see Section II.III.II). This work does not include the launch from the surface to a parking orbit in the trajectory model. Since most interplanetary trajectories are analyzed using a launch event, this is not a serious loss. The real purpose of including orbital departures in the trajectory model is to allow for round-trip scenarios which involve inserting into a parking orbit at a desired planet and then later departing to return home.

For a launch event, a launch vehicle will a feasible range on the hyperbolic escape velocity magnitude $V_\infty$. Therefore the required change in velocity is simply:

$$\Delta V_{\text{launch}} = V_\infty$$

(2.1)

However, in some cases, we may need to fully define the 3-dimensional velocity vector. Since the launch vehicle defines the magnitude, it makes reasonable sense to utilize a spherical coordinate system in which the hyperbolic escape velocity vector relative to the planet is:

$$V_\infty = \begin{bmatrix} V_\infty \\ \theta \\ \delta \end{bmatrix}$$

where $V_\infty$ is the magnitude of the velocity, and $\theta$ and $\delta$ are the right ascension and declination angles respectively. There is a well known problem with spherical coordinates [1] that if $\theta$ and $\delta$ are uniformly randomly sampled then the results will be a
biased towards the poles of the sphere. To overcome this bias, the variables $u$ and $v$ are introduced which are a transformation of $\theta$ and $\delta$ respectively:

\[
\begin{align*}
u &= \frac{\cos \delta + 1}{2} \\
\theta &= \frac{\theta}{2\pi}
\end{align*}
\]

The velocity can be calculated using the transformed angles as:

\[
V_{\infty} = V_\infty \begin{bmatrix} -\sin \delta \cos \theta \\ \sin \delta \sin \theta \\ -\cos \delta \end{bmatrix}
\]

In order to provide full 360 degree uniform sampling, the ranges for the transformed variables are simply:

\[
0 \leq u \leq 1 \\
0 \leq v \leq 1
\]

**II.III.II. Rendezvous**

Rendezvous marks the end of a interplanetary trajectory. The $\Delta V$ of a rendezvous event is simply the change in velocity necessary to match the arrival body's heliocentric velocity vector. This is similar to the $V_x$ of the launch event and is essentially an estimate of the $\Delta V$ required to land on the surface of the arrival body. The final change in velocity is calculated by taking the magnitude of the difference between the spacecraft's heliocentric approach velocity vector $V^-$ and arrival body's heliocentric velocity vector $V_{p,\text{arrive}}$:

\[
\Delta V_{\text{rendezvous}} = \|V^- - V_{p,\text{arrive}}\| 
\]
II.III.III. Orbital Departure

An orbital departure replaces the launch event if the spacecraft is starting from an initial parking orbit about the departure body. Orbital departures are not often used in this manner, but they can serve a much more useful alternative purpose: They allow round-trip scenarios in which a spacecraft may launch from a departure body and insert into a parking orbit about its target and then later depart from that orbit to return home. The required $\Delta V$ for an orbital departure is the difference between the spacecraft's velocity magnitude in parking orbit $V_{park}$ and the velocity magnitude at periapsis of the escape hyperbola $V_{phyp}$:

$$\Delta V_{depart} = \left| V_{phyp} - V_{park} \right|$$  \hspace{1cm} (2.3)

To calculate the velocity at the periapsis of the escape hyperbola, the first step is to determine the magnitude of the spacecraft's outbound relative velocity at infinity which is the magnitude of the difference between the spacecraft's desired outbound heliocentric velocity vector $\mathbf{v}^+$ and the planet's heliocentric velocity vector $\mathbf{v}_p$:

$$V_{\infty} = \left\| \mathbf{v}^+ - \mathbf{v}_p \right\|$$

The velocity at periapsis of the escape hyperbola is then calculated as:

$$V_{phyp} = \sqrt{V_{\infty}^2 + 2\mu R}$$  \hspace{1cm} (2.4)

where $\mu$ is the gravitational parameter of the departure body and $R$ is the distance (radius) of the spacecraft from the departure body's center calculated using the parking orbit's orbital elements:

$$R = \frac{h^2}{\mu}(1 + e \cos \theta)$$
The spacecraft's parking velocity is calculated as:

\[ V_{park} = \sqrt{\frac{\mu}{R}} (1 + e) \]  

(2.5)

II.III.IV. Orbital Insertion

An orbital insertion is an event in which the spacecraft transitions from a heliocentric elliptic orbit to a planetocentric elliptic orbit. Orbital insertions may be useful to space probes which are tasked with surveying a celestial body's surface or low altitude environment, or as a precursor to a surface landing. In order to model an orbital insertion, the orbital elements of the target orbit must be specified. In reality all six orbital elements would be required to fully characterize the orbit, however since this work is only interested in minimizing the total \( \Delta V \) of the mission, only two parameters are necessary to determine the insertion \( \Delta V \): the semimajor axis \( a \) and the eccentricity \( e \). The insertion into the desired parking orbit will occur at the periapsis of the spacecraft's hyperbolic trajectory as it flies past the planet. Therefore the insertion \( \Delta V \) is calculated similarly to the departure \( \Delta V \) and is the difference between the spacecraft's velocity at periapsis of the flyby hyperbola \( V_{phyp} \) and the spacecraft's velocity at periapsis of the parking orbit \( V_{park} \):

\[ \Delta V_{insert} = |V_{park} - V_{phyp}| \]  

(2.6)

The parking orbit periapsis velocity is calculated in the same manner as Equation (2.5). The spacecraft's velocity at periapsis of the flyby hyperbola is also calculated by first finding the magnitude of the spacecraft's incoming relative velocity at infinity which is
the magnitude of the difference between the spacecraft's inbound heliocentric velocity vector $V^-$ and the planet's heliocentric velocity vector $V_p$:

$$V_\infty = \|V^- - V_p\|$$

The velocity at periapsis of the flyby hyperbola is then the same as Equation (2.4).

II.III.V. Flyby

When a trajectory arc reaches an intermediate celestial body along its trajectory, there are two options for deciding how to handle the flyby event: The flyby can either be an unpowered or powered flyby. For an unpowered flyby, a spacecraft does not perform any maneuvering during the flyby. The change in energy the spacecraft experiences is only a result of the natural exchange between itself and the celestial body it is passing. For a powered flyby on the other hand, the spacecraft performs an $\Delta V$ during the periapsis of the inbound flyby hyperbola. The two approaches lead to two very different mathematical models. For this work, a unpowered flyby model was implemented for two primary reasons. For one, unpowered flybys are a more realistic representation of a flyby event. In reality, a spacecraft is unlikely to perform an instantaneous maneuver at the precise time of periapsis. More likely, is that a spacecraft will perform a corrective maneuver just before or just after the flyby which is more representative of the unpowered flyby. The other reason is that powered flybys impose additional nonlinear constraints on the problem whereas unpowered flybys can treat those constraints as state variables instead which will be discussed in later sections. For more information regarding powered flybys, refer to [1]. For an unpowered flyby, the spacecraft ($V^-$) approaches the planet ($V_p$) on a hyperbolic trajectory with a incoming relative velocity vector given by:
\[ V_\infty^- = V^- - V_p \]

It is known from the energy equation that, for a hyperbola, the inbound and outbound asymptotic velocities must be the same in magnitude:

\[ V_\infty = ||V_\infty^-|| = ||V_\infty^+|| \]

If the periapsis radius of the hyperbola is \( R_{PH} \), then the eccentricity of the hyperbola is given by:

\[ e_H = 1 + \frac{R_{PH}V_\infty^2}{\mu_p} \]

where \( \mu_p \) is the planet's gravitational parameter. The turn angle is the angle the incoming relative velocity vector is rotated by at the end of the hyperbolic trajectory and is defined as:

\[ \delta = 2\sin^{-1}\left(\frac{1}{e_H}\right) \]

Figure 2.2. Geometry of the flyby.
In two dimensional space, the outbound relative velocity would be a simple rotation of $\delta$ about the out-of-plane axis. However, to fully characterize the hyperbolic plane in three dimensions requires an additional parameter: The B-plane inclination angle $\xi$ (referred to as $b_{incl}$ elsewhere in this work). This is the angle of the hyperbolic plane relative to some reference plane. Using this third parameter, the outgoing relative velocity vector $\mathbf{V}_\infty^+$ can then be found using the equation [5]:

$$\mathbf{V}_\infty^+ = v_x \left[ \mathbf{\tilde{B}}_1 \cos(\delta) + \mathbf{\tilde{B}}_2 \cos(\xi) \sin(\delta) + \mathbf{\tilde{B}}_3 \sin(\xi) \sin(\delta) \right]$$  \hspace{1cm} (2.7)

where $\mathbf{\tilde{B}}_1$, $\mathbf{\tilde{B}}_2$, and $\mathbf{\tilde{B}}_3$ are unit vectors defined as:

$$\mathbf{\tilde{B}}_1 = \frac{\mathbf{V}_\infty}{\| \mathbf{V}_\infty \|}$$  \hspace{1cm} (2.8)

$$\mathbf{B}_2 = \mathbf{\tilde{B}}_1 \times \frac{\mathbf{V}_p}{\| \mathbf{V}_p \|}, \quad \mathbf{\tilde{B}}_2 = \frac{\mathbf{B}_2}{\| \mathbf{B}_2 \|}$$  \hspace{1cm} (2.9)

$$\mathbf{B}_3 = \mathbf{\tilde{B}}_1 \times \mathbf{\tilde{B}}_2, \quad \mathbf{\tilde{B}}_3 = \frac{\mathbf{B}_3}{\| \mathbf{B}_3 \|}$$  \hspace{1cm} (2.10)

For a more detailed explanation and derivation of these equations, refer to [1] and [5]. Note: equations 2.7-2.10 are presented slightly differently in the mentioned references, but the end result is the same. The way presented is based on the implementation in PaGMO (introduced in Section III.IV). Note, that since an unpowered flyby model is used, the change in velocity required at the flyby event is always zero:

$$\Delta V_{flyby} = 0$$  \hspace{1cm} (2.11)
II.III.VI. Deep Space Maneuver (DSM)

A deep space maneuver (DSM) is a change in velocity that occurs in deep space. The magnitude of a DSM is defined as the magnitude of the difference between the heliocentric velocity vectors before and after the maneuver:

$$\Delta V_{DSM} = \| V_f - V_i \|$$  \hspace{1cm} (2.12)

A DSM is a flexibility parameter which is used to connect conic arcs in deep space and can greatly increase the number and diversity of feasible solutions for a given problem.

II.IV. Velocity Formulation

Now that all the individual pieces of an interplanetary trajectory have been identified, they can be pieced together to form a complete interplanetary trajectory. There are two popular linked-conic approaches for doing so in the literature. The first is referred to as the velocity formulation whereas the second is the position formulation [1]. The fundamental difference between the two methods is how they handle the parameterization of DSMs (see Section II.IV.I below). The other major difference is that the velocity formulation uses unpowered flybys whereas position formulation uses powered flybys. This work chose to use the velocity formulation approach because unpowered flybys are more desirable for the reasons given in Section II.III.V. In the velocity formulation, the interplanetary trajectory is broken down into a series of planet-to-planet legs. If there are $n$ planets in the planetary sequence, then there will be $n-1$ legs. Each leg consists of a number of conic arcs depending on the number of DSMs that are allowed on that leg. For every $m$ allowed DSMs, there are $m+1$ conic arcs. The rules for deciding whether a conic
arc is a Lambert arc or propagation arc are simple: if the conic arc is connecting a launch or flyby event to a DSM, then it is a propagation arc, otherwise it is a Lambert arc.

\[
conic\ arc = \begin{cases} 
\text{launch} \rightarrow \text{DSM}, & \text{propagation arc} \\
\text{flyby} \rightarrow \text{DSM}, & \text{propagation arc} \\
\text{otherwise}, & \text{Lambert arc}
\end{cases}
\]

(2.13)

For example, consider a trajectory consisting of three planets with a DSM between the first and second planets. It is desired that the spacecraft launch from the first planet and rendezvous with the last planet. According to Equation 2.13, the arc connecting the planet 1 to the DSM is a propagation arc, the arc connecting the DSM to planet 2 is a Lambert arc, and the arc connecting planet 2 to planet 3 is a Lambert arc. Figure 2.3 displays the resulting trajectory.

Figure 2.3. Example interplanetary trajectory consisting of a launch, DSM, unpowered flyby, and rendezvous.
In order to calculate the total required change in velocity (mission velocity) of an interplanetary trajectory \( \Delta V_{\text{mission}} \) we need to sum up the \( \Delta V \) of each maneuver using Equations 2.1-2.3, 2.6, and 2.11-2.12:

\[
\Delta V_{\text{mission}} = \Delta V_{\text{launch}} + \sum \Delta V_{\text{flyby}} + \sum \Delta V_{\text{DSM}} + \sum \Delta V_{\text{depart}} + \sum \Delta V_{\text{insert}} + \Delta V_{\text{rendezvous}}
\]

Now a particular trajectory may not include all of these maneuvers in which case the \( \Delta V \) for that maneuver is simply zero. A common interplanetary trajectory is one that starts with a launch, ends with a rendezvous, and has \( n \) flybys in-between. This standard trajectory is referred to as a Multiple Gravity Assist (MGA) trajectory.

\[
\Delta V_{\text{mission}} \big|_{\text{MGA}} = \Delta V_{\text{launch}} + \sum_{i=1}^{n} \Delta V_{\text{flyby}, i} + \Delta V_{\text{rendezvous}}
\]

The number of feasible trajectories for a problem can typically be increased drastically by including a single DSM in between each planet. Therefore, with \( n \) legs, there will be \( n+1 \) DSMs. This type of trajectory is referred to as a MGA-1DSM trajectory.

\[
\Delta V_{\text{mission}} \big|_{\text{MGA-1DSM}} = \Delta V_{\text{launch}} + \sum_{i=1}^{n} \Delta V_{\text{flyby}, i} + \sum_{j=1}^{(n+1)} \Delta V_{\text{DSM}, j} + \Delta V_{\text{rendezvous}}
\]

More generally, if \( m \) DSMs are allowed for each leg, then the trajectory is referred to as a MGA-DSM trajectory.

\[
\Delta V_{\text{mission}} \big|_{\text{MGA-DSM}} = \Delta V_{\text{launch}} + \sum_{i=1}^{n} \Delta V_{\text{flyby}, i} + \sum_{j=1}^{(n+1)+m} \Delta V_{\text{DSM}, j} + \Delta V_{\text{rendezvous}}
\]

When there is only one DSM per leg, the terms MGA-1DSM and MGA-DSM are used interchangeably in this work, the latter is just more general. In order to calculate the
\( \Delta V_{\text{mission}} \) for a particular trajectory type, we first need a way of parameterizing the trajectory.

II.IV.I. **Trajectory Parameterization**

In order to calculate an interplanetary trajectory of a particular type, we first need to establish the minimum set of parameters that fully characterizes all of the conic arcs and maneuvers involved in the trajectory. Let's start with the launch event. The first thing we need to know in order to start a trajectory is the launch date and time, because without them, we have no way of determining the initial heliocentric position and velocity of the departure body. Dates use many different formats, such as the Gregorian date system which is the Month/Day/Year format we are most likely familiar with, but the Gregorian system is not easy to work with in mathematics because it requires 3 numbers to represent one date and because the amount of time in-between two dates is not immediately obvious. A much more useful system is the Julian date (JD) system. A Julian day is the number of days since January 1st, 4713 BC Greenwich noon. The Julian day system is especially useful because it incorporates time as well. Time is represented as a fraction of one Julian day where 0.5 is Greenwich noon. So the Gregorian date January 1st, 2011 AD at midnight is the Julian day 2455563.5. The JD system is a step in the right direction, but one drawback is modern Gregorian dates are quite large Julian dates (8 digits). So in the late 1950's the Modified Julian date (MJD) was introduced which is the number of days since November 17th, 1858 at midnight. That may seem like an arbitrary date, but it makes converting between JD and MJD very convenient:

\[
MJD = JD + 2400000.5
\]
The result is that modern Gregorian dates are now only 5 digits in MJD format. For example, the Julian day 2455563.5 from earlier is now the Modified Julian day 55562. This work makes use of yet another modified version of the JD system called MJD2000 which is the number of days since January 1st, 2000 at midnight. Converting between MJD and MJD2000 is:

\[ MJD2000 = MJD + 51544 \]

Therefore our example MJD day of 55562 is now the MJD2000 day 4018. The modified Julian dates don't alter any of the mathematics, they simply make the numbers more convenient to work with. In addition to the launch date, a launch event always requires the excess velocity magnitude \( V_x \) and may also require the normalized angle parameters \( u \) and \( v \) in order to define the 3d velocity vector if the launch is followed by a DSM. Therefore the launch event can be completely characterized by the vector:

\[ \mathbf{x}_{\text{launch}} = [MJD2000, V_x, u, v] \]  

(2.14)

This vector is called the state vector. As we incorporate more maneuvers into the mission, the state vector will grow with the additional parameters required to fully characterize the increasingly complex trajectory.

Following the launch event, the next maneuver is a DSM (if applicable). Since we know from Equation 2.13 that the conic arc between a launch and a DSM event is a propagation arc, and the launch fully described our initial heliocentric position and velocity vectors, the only parameter left is the propagation time of flight (TOF). However, specifying a TOF for a DSM is awkward, it is much more intuitive to define a TOF for the entire leg. Then, the TOF of the DSM can simply be expressed as a fraction of the entire TOF of the
leg. Therefore the only parameter we need for the DSM is $\alpha$ which is a fractional number between zero and one (inclusive):

$$x_{DSM} = [\alpha] \quad 0 \leq \alpha \leq 1$$

$TOF_{propagation} = \alpha TOF_{leg}$

The TOF of the leg is not included here, because it makes better sense to include it with the flyby/orbital insertion/rendezvous state vectors as described below. At the end of the propagation arc, we have a final heliocentric position and velocity somewhere in deep space.

After the DSM will typically be a flyby event. From equation 2.13 we know the conic arc connecting a DSM to a flyby is a Lambert arc. The Lambert arc requires the initial heliocentric position (at the DSM) and the final heliocentric position (at the flyby) along with the TOF. We already know the position at the DSM, and now is a good time to specify the TOF of the leg which will define the flyby planet's heliocentric position (using analytic/ephemeris propagation) and the remaining TOF between the DSM and the flyby planet after subtracting off the TOF of the DSM:

$$TOF_{Lambert} = (1 - \alpha)TOF_{leg}$$

At this point, we can calculate the Lambert arc which gives us the spacecraft's heliocentric approach velocity at the flyby planet. The next step is using the unpowered flyby model (see Section II.III.V) to calculate what the spacecraft's velocity vector will be after flyby. To do so, we need two more parameters: the minimum allowable hyperbola periapsis radius $r_p$ (or flyby altitude) and the $B$-plane inclination angle $b_{incl}$. Therefore the state vector for the flyby event is:

$$x_{flyby} = [T, r_p, b_{incl}] \quad -\pi \leq b_{incl} \leq \pi$$
Following the flyby, if the next maneuver is a DSM, then we will use a propagation arc again, otherwise we will be using a Lambert arc to connect to the next planet. Eventually, we will reach the end of the trajectory with a rendezvous maneuver. The rendezvous maneuver is similar to the flyby maneuver except (since we're not flying by) we don’t need to use the unpowered flyby method. Therefore the state vector for the rendezvous is simply the leg TOF:

\[ \mathbf{x}_{\text{rendezvous}} = [T] \]  

(2.17)

Using equations 2.14-2.17 we now have all the state vectors defined to specify any MGA or MGA-1DSM trajectory (excluding round-trip missions for the moment). The full state vector for a MGA trajectory with \( n \) legs is:

\[
\mathbf{x} = [MJD2000, V_x, \]

\[
T_1, r_{p1}, b_{incl1},
\]

\[
, \ldots,
\]

\[
T_{n-1}, r_{pn-1}, b_{incln-1},
\]

\[
T_n]
\]

The full state vector for a MGA-1DSM trajectory with \( n \) legs is:

\[
\mathbf{x} = [MJD2000, V_x, u, v, \]

\[
\alpha_1, T_1, r_{p1}, b_{incl1},
\]

\[
, \ldots,
\]

\[
\alpha_{n-1}, T_{n-1}, r_{pn-1}, b_{incln-1},
\]

\[
\alpha_n, T_n]
\]

The state vectors from Equations 2.17 and 2.18 are good, but we left out a few possibilities, namely round-trip scenarios using orbital insertions/departures and multiple
DSMs per leg. In the case of the former, a round-trip scenario involves a orbital insertion/departure at a target planet, referred to hereafter as a *surface stay*, which replaces an unpowered flyby. Inserting into an orbit about a target planet is achieved by specifying the eccentricity and semimajor axis of the desired parking orbit which are constants and thus not included in the state vector (the state vector represents the *variables* of the trajectory which we will later be attempting to optimize). Once in the parking orbit, we need to specify how long we want to orbit the target planet before departing, i.e. the surface stay time $T_{stay}$. Similar to the launch event, if the surface stay is followed by a DSM then we need to fully define our departure velocity vector so that a propagation arc can be used. Otherwise, a Lambert arc is used and we only need the magnitude of the velocity. The state vector for the surface stay event is:

$$x_{surface\_stay} = [T, T_{stay}, V_x, u, v]$$ (2.19)

Using equation 2.19, the full state vector for a MGA-1DSM trajectory with $n$ legs and a surface stay replacing the $i$'th flyby is then:

$$x = [MJD2000, V_x, u, v, \\alpha_1, T_1, r_{p1}, b_{incl1}, \\ldots, \\alpha_i, T_i, T_{stay}, V_x, u, v, \\ldots, \\alpha_{n-1}, T_{n-1}, r_{pn-1}, b_{incln-1}, \\alpha_n, T_n]$$ (2.20)
The last order of business is to handle multiple DSMs on a single leg. In the literature, the capability for multiple DSMs on a single leg is usually not supported. This work chose to include them for completeness, but in hindsight, their usefulness is questionable. Whereas the first DSM that is added to a leg tends to drastically increase the number of feasible trajectories and improves diversity, additional DSMs tend to have a diminishing effect. To make matters worse, additional DSMs past the first one greatly increase the size of the state vector. The conic arc connecting a planet to a DSM is always a propagation arc and after the propagation we know the final heliocentric position and velocity at the DSM. In order to connect an arc to another DSM, we need to either define the 3d \( \Delta V \) velocity vector at the first DSM and use another propagation arc, or specify the 3d position vector of the second DSM and use a Lambert arc. This work chose the former because it is more in line with the velocity formulation. Therefore, in order to add a second DSM to a leg, we must add three parameters defining the \( \Delta V \) velocity vector (similar to the launch and orbital departure events) along with another \( \alpha \) parameter which indirectly specifies the TOF between the two DSMs. The resulting state vector is:

\[
x_{DSM^+} = [\alpha, \Delta V_{DSM}, u, v]
\]

When dealing with multiple \( \alpha \)'s for a single leg, the first propagation TOF is calculated based on the TOF of the entire leg:

\[
TOF_{prop,1} = \alpha_1 TOF_{leg}
\]

\[
TOF_{remaining} = TOF_{leg} - TOF_{prop,1}
\]

and the subsequent TOF's are calculated based on the TOF remaining:

\[
TOF_{prop,i} = \alpha_i TOF_{leg,remaining}
\]

\[
TOF_{remaining} = TOF_{remaining} - TOF_{prop,i}
\]
Doing so in this manner prevents the need to enforce that subsequent $\alpha$'s are always larger than their predecessors (otherwise a negative TOF would result).

In conclusion, a MGA-DSM trajectory with $n$ legs and $m+1$ DSM's on each leg, is defined by the full state vector:

$$\mathbf{x} = [V_x, u, v, \alpha_{11}, \alpha_{12}, u_{12}, \Delta V_{dsm12}, ..., u_{1m}, v_{1m}, \Delta V_{dsm1m}, T_1, r_{p1}, b_{incl1},$$

$$\alpha_{n-1,1}, \alpha_{n-1,2}, u_{n-1,2}, \Delta V_{dsmn-1,2}, ..., u_{n-1,m}, v_{n-1,m}, \Delta V_{dsmn-1,m}, T_{n-1}, r_{pn-1}, b_{incln-1},$$

$$\alpha_n, T_n]$$

Note the drastic increase in complexity when additional DSMs are added to each leg. The first DSM of each leg adds only one state variable $\alpha$, however adding more DSMs to that leg adds an additional four state variables $\alpha, u, v, \Delta V_{dsm}$.
III. Global Optimization

This chapter will give an introduction to the field of global optimization. This chapter first breaks down what a global optimization problem is, then discusses the challenges in solving such problems, and lastly overviews different methods that are typically used to solve them. This chapter also introduces PaGMO which is a software implementation of the Island Model Paradigm created by the European Space Agency which is fundamental to this work, and the GTOP Database which is a standardized set of MGA-1DSM problems that provide a test bed for methods explored in this work.

III.I. Introduction

A global optimization problem is typically of the form [5]:

\[
\begin{align*}
\text{find:} & \quad x \in \mathbb{R}^n \\
\text{to minimize:} & \quad f(x) \\
\text{subject to:} & \quad g(x)
\end{align*}
\]  

(3.1)

where \( x \) is decision or state vector, \( f(x) \) is the objective or cost function, and \( g(x) \) are any nonlinear constraints on the problem. The objective function and constraints \( f \) and \( g \) are vectors because generally there can be more than one objective or constraint active. Problems that deal with more than one objective function are multi-objective optimization problems and at the time of writing are not supported in this work. The nonlinear constraints are typically additional restrictions or requirements that are placed on the model, but not accounted for in the model. A nonlinear constraint may be very similar to
the objective function, or it could have nothing to do with it at all. For example, the objective function used in this work is the minimization of the total mission delta-v:

\[ f(\mathbf{x}) = \Delta V_o + \sum_{i=1}^{n} \Delta V_i + \Delta V_f \]  

(3.2)

where \( \Delta V_o \) is the departure delta-v, \( \Delta V_i \) is the delta-v associated with the \( i \)'th DSM, and \( \Delta V_f \) is the arrival delta-v. Two common nonlinear constraints that could be placed on this problem are the max mission delta-v and the max mission time of flight:

\[
\Delta V_{mission}(\mathbf{x}) \leq \Delta V_{mission}^{max} \\
TOF_{mission}(\mathbf{x}) \leq TOF_{mission}^{max}
\]  

(3.3)

All solutions have an \textit{objective function value} or \textit{cost} according to (3.2), but nonlinear constraints such as (3.3) place additional burden on the optimizer to "do something special" with the solutions that are invalid, i.e. don't satisfy all of the constraints. This realm of optimization is referred to as \textit{constraint optimization}. This work avoids constraint optimization by using the velocity formulation trajectory model (see Chapter II) which doesn't innately require any additional nonlinear constraints. Constraints such as (3.3) could still be applied in addition to the outlined trajectory model, but at the time of writing, this work does not support them. Therefore, all global optimization problems in this work are reduced to the simplified form:

\[ \text{find: } \mathbf{x} \in \mathbb{R}^n \]  

(3.4)

\[ \text{to minimize: } f(\mathbf{x}) \]
III.II. The Search Space

For a global optimization problem, the terms search space, search domain, and feasible region are commonly used to represent the boundaries in which all possible solutions to the problem are contained within. For example, consider a global optimization with \( n \) variables \( x_1, x_2, ..., x_i, ..., x_n \) where each variable is bounded by minimum and maximum values \( a_i \) and \( b_i \) respectively.

\[
[a_1 < x_1 < b_1], [a_2 < x_2 < b_2], ..., [a_i < x_i < b_i], ..., [a_n < x_n < b_n]
\]

The search space \( D \) is then the Cartesian product of all \( n \) bounding intervals.

\[
D = [a_1, b_1] \times [a_2, b_2] \times ..., \times [a_i, b_i] \times ..., \times [a_n, b_n]
\]

All feasible solutions to the problem are therefore contained within \( D \). In vector form, we have

\[
D = \{ \mathbf{x} \in \mathbb{R}^n | \mathbf{a} \leq \mathbf{x} \leq \mathbf{b} \}
\]

Where \( \mathbf{a} \) and \( \mathbf{b} \) are the vectors of lower and upper bounds respectively. It is then natural to think of solving a global optimization problem as "searching" for the best solution within the search space. More specifically, we are concerned with finding the optimal \( \mathbf{x} \) within \( D \) which minimizes the cost function \( f(\mathbf{x}) \). As such, every \( n \)-dimensional point in \( D \) has an associated cost. In two dimensions, this is easily visualized using a contour or surface plot. Take for example, the popular Peaks function in MATLAB characterized by

\[
\text{peaks}(x_1, x_2) = 3(1 - x_1)^2e^{-(x_1^2 - (x_2 + 1)^2)} - 10\left(\frac{x_1}{5} - x_1^3 - x_2^5\right)e^{-(x_1^2 - x_2^2)} - \frac{1}{3}e^{-(x_1 + 1)^2 - x_2^2}
\]

Let's say we wanted to find the \( x_1 \) and \( x_2 \) pair which minimizes the Peaks function over the ranges
\[-3 \leq x_1 \leq 3 \text{ and } -3 \leq x_2 \leq 3\]

Therefore our cost function is the Peaks function itself.

\[f(x) = \text{peaks}(x)\]

We could have chosen to maximize the Peaks function instead in which case our cost function would simply be negated.

\[f'(x) = -\text{peaks}(x)\]

A filled contour plot and surface plot are shown in Figure 3.1. We can think of the search space as the projection onto the \(x_1-x_2\) plane. It is also useful to think of the surface plot of the cost function as the \textit{fitness landscape}. As we move through the search space, we are essentially exploring the fitness landscape with hopes of identifying the \textit{best fit} solution, which is the solution that minimizes our cost function. Since we want to find the solution which minimizes the Peaks function, the best fit solution corresponds to the global minima. If we had wanted to maximize the Peaks function, the best fit solution would then correspond to the global maxima.
Even though the Peaks function is a simple two dimensional example, the same ideas apply to problems of any dimension. In two dimensions, the search space is visualized as a rectangle characterized by the $x_1$-$x_2$ plane. More generally, the search space of a $n$-dimensional problem can be visualized as a $n$-dimensional hyperrectangle characterized by the $x_1$-$x_2$-$...$-$x_n$ plane. In three dimensions, this would correspond to a cube. The fitness landscape then, is always a $n+1$ dimensional surface.

The Peaks function is a nice example of a well behaved fitness landscape, but unfortunately, this may not always be the case for other problems. A fitness landscape can be characterized by how smooth, modal, and noisy, it is. The smoother a landscape is, the fewer discontinuities it exhibits at a macroscopic level, i.e, from far away. Discontinuities are generally disruptive to global optimizers because they cause them to become disoriented or make it very hard to fully converge to a local optima. How modal a landscape is refers to how many local optima exists. Landscapes which are unimodal
have only one local optima which is also the global optima. Multimodal landscapes are those with many local optima. It is generally much more difficult to find the global optima from a very multimodal cost function because an optimizer may get "stuck" at a local optima (also known as premature convergence). Local optimizers are especially prone to premature convergence to local optima due to their intrinsic intensity-driven nature as compared to global optimizers which promote more diversity. Lastly, the noise of a landscape refers to how continuous it is at a microscopic level. To illustrate these characteristics, consider for example, Rastrigin's function in two dimensions

\[ Ras(x) = f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2) \]

over the range

\[-3 \leq x_1 \leq 3 \text{ and } -3 \leq x_2 \leq 3\]

Figure 3.2 shows a contour and surface plot. Here the landscape would be classified as smooth and quiet because it is looks continuous from far away as well as up close. The function is multimodal because there are noticeably many local optima.

![Figure 3.2. Rastrigin's function demonstrating a highly multimodal fitness landscape.](image)
Below is Rastrigin’s function again, but this time with an additional term \( \text{rand}(-n,n) \) which generates random numbers between \(-n\) and \(n\). This additional term essentially produces noise.

\[
Ras'(x) = f'(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2) + \text{rand}(-n,n)
\]

The search space is reduced in order to better see the affect of the noise.

\[-1 \leq x_1 \leq -1 \text{ and } -1 \leq x_2 \leq 1\]

Figure 3.3 shows two resulting surface plots, one with \(n=5\) and one with \(n=20\). The function is still considered smooth, but the affect of the noise is obviously apparent as \(n\) increases.

Figure 3.3. Rastrigin's function with added noise. Left: \(n=5\). Right: \(n=20\).
MGA trajectories typically have smooth, multimodal, and noisy fitness landscapes. In addition, allowing a single DSM on each planetary arc drastically increases the modality and noise of the fitness landscape. Thus, MGA-DSM problems are extremely difficult to solve even though their dimensionality may be much smaller compared to other problems in engineering which may have hundreds or even thousands of variables.

III.III. Algorithms

Solving global optimization problems of the form (3.1) or (3.4) are an immense field of applied mathematics. There are numerous algorithms for solving such problems, but the most popular are loosely subdivided into three general categories: Deterministic Methods, Calculus of Variations, and Metaheuristics.

**Deterministic Methods**

Deterministic methods are those that solve a problem in a systematic manner. Such methods are non-probabilistic which means that every trial produces the same result because each trial proceeds in exactly the same manner. Deterministic methods are typically forms of brute force algorithms which systematically march through the search space in finite steps, exploring all possible combinations of input variables. Some deterministic methods are more sophisticated, such as the DIRECT (Divided Rectangles) algorithm, and aim to allocate most of their resources in areas of the search space that appear to have more potential for favorable solutions than others. Ultimately though, the quality of a deterministic method is dependent on how much it discretizes each variable, i.e. its fidelity. A deterministic method with a high fidelity (discrete steps are very small)
can solve low dimensional problems to a high degree of accuracy fairly quickly and easily. However, discretization in this manner leads to exponential complexity. As more dimensions are added to the problem, the amount of time needed to perform a search of constant fidelity increases exponentially. This effect is commonly referred to as *The Curse of Dimensionality*. Therefore deterministic methods are efficient and effective with problems with less than three dimensions, but began to require unreasonable computation time past five or six dimensions. Thus, they are ill-suited for MGA-DSM problems which are typically on the order of 12-26 dimensions.

**Calculus of Variations**

Calculus of Variations (CoV) is a large subfield of GO which is based on the idea of using the derivative of the objective function to help find optimal solutions. Such methods are commonly referred to as *gradient-based searches*, because the derivative provides information about the slope or gradient at a particular point in the fitness landscape. Calculus of Variations methods are usually classified as either Direct or Indirect methods. In addition, Optimal Control Theory is a extension of Calculus of Variations which is commonly applied to trajectory optimization. Methods in this category are usually very effective in problems with low modality, i.e. few local optima, however they tend to struggle with very multimodal fitness landscapes. CoV methods can be applied to MGA-DSM problems, but this work chose to concentrate on metaheuristics.
Metaheuristics

Metaheuristics is another huge subfield of GO. The exact definition tends to vary widely, but all definitions agree (as the name suggests) that metaheuristics make use of heuristics. A heuristic is a strategy which aims at solving a problem reasonably well in a reasonable amount of time. A heuristic doesn't guarantee global convergence, nor can it even prove that the proposed solution is in fact the global minima. It makes these sacrifices in order to reduce the computation time, sometimes drastically. Good heuristics can typically find optimal or nearly optimal solutions most of the time and in much less time than it would have taken an exhaustive search to identify a solution of similar caliber.

A metaheuristic is a higher level algorithm which, unlike a heuristic, is usually problem-independent. This means metaheuristics don't usually know anything about the problem they are trying to solve, which makes them highly generalized and useful for a variety of different optimization problems. Metaheuristics rely on black-box functions, which are typically heuristics, to iteratively improve one or multiple candidate solutions. Like heuristics, metaheuristics cannot guarantee global convergence, and tend to have stochastic mannerisms. Solving global optimization problems using metaheuristics is particularly difficult because algorithms constantly have to appeal to two very different and often conflicting goals: diversification and intensification. Diversification and intensification refer to how thoroughly the search space is explored globally and locally respectively [29]. Algorithms that place too much emphasis on diversity, may only have time to skim the surface of the fitness landscape because they must cover a large amount of ground very quickly. As a result, difficult solutions with narrow basins of attraction will be very difficult to find because so little effort is invested in searching a local area
long enough to find them. On the flip side, intensity driven algorithms may do a superb
job finding those solutions hiding at the bottom of narrow basin of attraction, but only if
it is fortunate to start its search nearby. As a loose rule of thumb, an algorithm that
repeatedly finds good solutions, but rarely finds great ones, could be suffering from too
much diversity whereas an algorithm that is very hit-or-miss is probably too intensity
oriented. Such is the battle that every metaheuristic must wage. It just isn't possible for a
metaheuristic to be perfect in both aspects without essentially becoming an exhaustive
deterministic method. Metaheuristics can be broken down into many sub-categories, but
the most general are Stoachstic Algorithms and Evolutionary Algorithms.

**Stochastic Algorithms**

A Stochastic method relies on randomness and probability to explore a feasible region.
Typical strategies involve perturbing a candidate solution in a random and probabilistic
manner and comparing the resulting quality of the new solution to the existing one.
Popular stochastic methods are Monte Carlo simulations, Basin-Hopping, and Simulated
Annealing. Of those mentioned, Simulated Annealing (SA) has proven to be very
effective in solving MGA-DSM trajectories [17]. Simulated annealing was inspired by
statistical mechanics and annealing in metallurgy. The algorithm works by iteratively
perturbing each state variable of the candidate solution according to a PDF (e.g.
uniform or Gaussian) distribution and a step size: the larger the step size, the larger the
allowed perturbation. After each perturbation, the resulting solution is compared to the
original one and if an improvement was made then it replaces the original. If the new
solution is not better than the original, then it still has a chance to replace it (an *up-hill* move) according to the Boltzmann probability:

\[ p_{accept} = \exp\left( -\frac{|f_{old} - f_{new}|}{T} \right) \]

where \( f_{old} \) and \( f_{new} \) are the old and new objective functions respectively and \( T \) is the temperature. The temperature is a parameter which decreases or "cools" according to a *temperature schedule* as the optimization progresses, which is where SA gets its name: In the annealing process in metallurgy, a material is heated and cooled in order to reduce its number of defects by dislodging atoms and allowing them to find better configurations (states of lower internal energy). The algorithm follows the same idea: A high initial temperature allows solutions to move up-hill (dislodge from a local optima) and as the temperature cools, the probability of moving up-hill decreases so that eventually the candidate solution is forced to settle to a new (and hopefully better) local optima. This work makes use of a variant of SA called Adaptive Simulated Annealing (ASA) as proposed by [10]. The difference is this algorithm adaptively adjusts its step size for each state variable depending on how "active" that variable is. If perturbating a particular state variable is resulting in many improvements, then that state variable is considered very active and the step size for that variable is increased, and vice versa. Pseudocode for the algorithm is outlined below:
Adaptive Simulated Annealing (ASA) Pseudocode

1. \( x_{\text{old}} \leftarrow x_0; \quad f_{\text{old}} \leftarrow f_0; \quad T = T_0; \quad \text{step} = \text{step}_0; \quad s = 0 \)
2. \textbf{for} iter \(<\) iter\(_{\text{max}}\)
3. \textbf{for} \(i \leftarrow 1:n_{\text{step}}\)
4. \textbf{for} \(j \leftarrow 1:n_{\text{bin}}\)
5. \textbf{for} \(k \leftarrow 1:n_{\text{dim}}\)
6. \( x_{\text{new}} \leftarrow \text{perturbate}(x_{\text{old}}, \text{step}_k) \)
7. \( f_{\text{new}} \leftarrow \text{objFun}(x_{\text{new}}) \)
8. \textbf{if} \( f_{\text{new}} < f_{\text{old}} \) \text{OR} \( \text{rand}(0,1) < \text{Boltzmann}(f_{\text{old}}, f_{\text{new}}, T) \) \textbf{then}
9. \( x_{\text{old}} \leftarrow x_{\text{new}}; \quad f_{\text{old}} \leftarrow f_{\text{new}}; \quad s_k \leftarrow s_k + 1 \)
10. \textbf{if} \( f_{\text{new}} < f_{\text{best}} \) \textbf{then}
11. \( x_{\text{best}} \leftarrow x_{\text{new}}; \quad f_{\text{best}} \leftarrow f_{\text{new}} \)
12. \( k \leftarrow k + 1 \)
13. \( j \leftarrow j + 1 \)
14. \( \text{step} \leftarrow \text{adjustStepSize}(s, \text{step}) \)
15. \( s \leftarrow 0 \)
16. \( i \leftarrow i + 1 \)
17. \( T \leftarrow \text{coolingSchedule}(T) \)
18. \( \text{iter} \leftarrow \text{iter} + 1 \)
19. \textbf{return} \([x_{\text{best}}, f_{\text{best}}]\)

Here \( x \) and \( f \) are the state vector and objective function respectively, \( T \) is the temperature, \( \text{step} \) is the vector of step sizes for each state variable, and \( s \) is the "activity" vector for
each state variable. In addition, \( n_{\text{dim}} \) is the number of dimensions of the state vector, \( n_{\text{bin}} \) is the bin size (the number of full perturbation cycles before changing the step size), \( n_{\text{step}} \) is the number of times the step size is adjusted, and \( n_{\text{iter}} \) is the number of times the temperature is cooled.

**Evolutionary Algorithms**

Evolutionary Algorithms is subfield of metaheuristics which traditionally centered around the idea of evolving a population of candidate solutions. Each iteration of the evolution, or generation, results in an increase in the overall quality of the population. Therefore, through repeated generations, the process inevitably leads to local optima. Originally, these algorithms were predominantly inspired by the biological mechanism of evolution (hence the name), although now evolutionary algorithms encompass a much wider variety of flavors. One of the first popular evolutionary algorithms, and one that is still heavily used today, is the Genetic Algorithm [7]. A genetic algorithm works by treating the candidate solutions as individuals who iteratively breed children solutions. A survival of the fittest mentality is in play which forces children to compete with each other and with their parents in order to maintain their spot in the population and become parents of the next generation. Breeding is accomplished through a process of selection, cross-over, and mutation and is widely open-ended. The genetic algorithm has inspired countless other algorithms, most notably of which is Differential Evolution [8]. Differential evolution (DE) is based on a simple concept and yet has proven to be one of the most versatile and powerful metaheuristics created. Differential evolution treats the candidate solutions as agents and iteratively perturbates each agent in the population by
adding to it the weighted vector difference between two (or more) other agents. There are many variants of DE which determine the number and manner in which agents are chosen from the population to calculate the weighted vector difference. Below is the pseudocode for differential evolution:

**Differential Evolution (DE) Pseudocode**

1. \( \text{pop} \leftarrow \text{randomPop}\) (
2.  \( \text{for} \ \text{iter} \leftarrow 1: n_{\text{gen}} \)
3.  \( \text{for} \ i \leftarrow 1: n_{\text{pop}} \)
4.  \( \text{do} \)
5.  \( r_1 \leftarrow \text{rand}(1,n_{\text{pop}}); \ r_2 \leftarrow \text{rand}(1,n_{\text{pop}}); \ r_3 \leftarrow \text{rand}(1,n_{\text{pop}}) \)
6.  \( \text{while} \ (\text{notUnique}(i,r_1,r_2,r_3)) \)
7.  \( x_{\text{inew}} \leftarrow x_{r_1} + m \ast (x_{r_2} - x_{r_3}) \)
8.  \( f_{\text{inew}} \leftarrow \text{objFun}(x_{\text{inew}}) \)
9.  \( \text{if} \ f_{\text{inew}} < f_i \ \text{then} \)
10. \( x_i \leftarrow x_{\text{inew}}; \ f_i \leftarrow f_{\text{inew}} \)
11. \( \text{if} \ f_{\text{inew}} < f_{\text{best}} \ \text{then} \)
12. \( x_{\text{best}} \leftarrow x_{\text{inew}}; \ f_{\text{best}} \leftarrow f_{\text{inew}} \)
13. \( i \leftarrow i + 1 \)
14. \( \text{return}(x_{\text{best}}, f_{\text{best}}) \)
Here, $pop$ is the population of agents and $x_i$ and $f_i$ are the state vector and objective function of each agent within $pop$. In addition, $m$ is the weighting factor (between 0 and 1), $n_{gen}$ is the number of generations and $n_{pop}$ is the population size. Each agent in $pop$, is pertubated by the weighted vector difference of three other randomly selected agents with indexes $r1$, $r2$, and $r3$. 

More recently, the focus of evolutionary algorithms have shifted to swarm-based optimization algorithms. The roots of swarm-based optimization were in the early efforts of simulating the swarm and flocking behavior of animals and insects observed in nature. It was realized eventually that swarm-based simulations could be effective when applied to global optimization and swarm-based optimization was born. Popular swarm-based algorithms include Particle Swarm Optimization, Ant Colony Optimization, Cuckoo Search, and Firefly Algorithm. Of the swarm-based optimizers, particle swarm optimization (PSO) [9] is one of the most common and has shown good success on MGA-DSM problems [6]. Particle swarm optimization treats the candidate solutions as particles where each particle has a position (state vector) and velocity (perturbation vector). The particles move throughout the search space according to their velocity which is iteratively adjusted based on inertial, cognitive, and social influences. Each iteration, each particle's velocity vector is updated using the particle's previous velocity vector (inertia), the position of the best solution the particle has personally discovered (cognitive), and the position of the best known solution the entire swarm has discovered (social). As with the previous algorithms, there are many many variations on PSO including using sub-swarms, collision detection, etc. Below is general pseudocode for PSO:
Particle Swarm Optimization (PSO) Pseudocode

1. pop ← randomPop()
2. for iter ← 1: n_gen
3. for i ← 1: n_pop
4.     r_1 = rand(0,1);  r_2 = rand(0,1)
5.     v_i ← ω * v_i + r_1 * η_1 * (x_{pbest,i} - x_i) + r_2 * η_2 * (x_{gbest,i} - x_i)
6.     x_{inew} ← x_i + v_i
7.     f_{inew} ← objFun(x_{inew})
8.     if f_{inew} < f_i then
9.         x_i ← x_{inew};  f_i ← f_{inew}
10.    if f_{inew} < f_{pbest,i} then
11.         x_{pbest,i} ← x_{inew};  f_{pbest,i} ← f_{inew}
12.    if f_{inew} < f_{gbest} then
13.         x_{gbest} ← x_{inew};  f_{gbest} ← f_{inew}
14.     i ← i + 1
15. iter ← iter + 1
16. return(x_{gbest}, f_{gbest})

Here, as usual, pop is the population containing the particles. The state vector, velocity vector, and objective function of each particle in pop are x_i, v_i, and f_i respectively. The parameters ω, η_1, and η_2 are the inertial, cognitive, and social weights respectively and r_1 and r_2 are two randomly selected numbers between 0 and 1. Lastly, n_gen is the number of generations and n_pop is the population size.
3.2 The Island Model & PaGMO

The metaheuristic algorithms mentioned above in Section III.I typically perform well on simple MGA-DSM problems, but tend to lose effectiveness as trajectories become more complex: frequently converging prematurely to local optima. More importantly, metaheuristics tend to be problem dependent. Differential evolution and particle swarm optimization, may outperform each other depending on the problem at hand. Determining a priori which algorithm is best suited for a given problem is not always trivial. There have been many proposed solutions to this problem ranging from full integration of different algorithms into one (hybrids) to full parallelization of algorithms working simultaneously. This work explores the latter option using a strategy called the Island Model Paradigm [15],[17]. The island model paradigm attempts to overcome the shortcomings of individual GO algorithms by combining them in parallel to allow them to work together and feed off each other's strengths. The basic idea is to start with a central population and break it into multiple sub-populations. Each sub-population is then assigned to an island along with a particular GO algorithm. Islands are then allowed to evolve simultaneously, but completely independently, using their own respective algorithm. The islands remain completely independent until a migration is allowed to occur. During migration, an island selects one or multiple solutions from its population according to a Selection Policy and distributes that solution to the other islands that are "connected" to it. Islands that receive a solution(s) from another island decide whether to replace solutions currently in their population with the newcomers or reject them entirely. These decisions are made according to an island's Replacement Policy. In addition, the number of islands used and the connections between islands is customizable and is
referred to as the *Island Topology*. The full configuration of a set of islands is often referred to as an *Archipelago* which literally means "group of islands." The archipelago migrations can either be synchronous or asynchronous. A synchronized scheme enforces that all migrations occur at the same time, whereas an asynchronous scheme allows islands to send and receive solutions at their own leisure. It's been found [17] that asynchronous schemes tend to outperform their synchronized counterparts.

The European Space Agency's (ESA) Advanced Concepts Team (ACT) developed a particularly useful implementation of the Asynchronous Island Model called PaGMO (Parallel Global Multiobjective Optimiser) [14-17]. PaGMO is a cross-platform open-source application written in object-oriented C++, whose goal is to provide a testbed infrastructure for testing global optimizers on various problems. PaGMO is a general optimization tool and is not restricted to solving MGA-DSM problems, but it pays particular interest to MGA and MGA-DSM problems. A user is free to construct an archipelago using the various problems and algorithms supplied by PaGMO, and can also create custom problems or algorithms by writing their own code. PaGMO also has a sister project named PyGMO which provides a Python wrapper around the C++ code. PyGMO allows the user to interact with a Python shell and create archipelagos interactively which is much more user-friendly. This work uses PaGMO as the underlying optimization engine and explores additional ad-hoc methods for improving its performance and capabilities with respect to MGA-DSM problems. At the time of writing, PaGMO's infrastructure also supports constraint optimization and even multiobjective problems, however these features were not used in this work.
3.3 GTOP Database

The European Space Agency made another major contribution to the global optimization community by creating the GTOP (Global Trajectory Optimization Problem) Database [18]. The GTOP Database is a standardized suite of challenging spacecraft interplanetary trajectory optimization problems. Its purpose is to allow global optimization enthusiasts and researchers to test their algorithms against a common set of difficult problems. The GTOP Database consists of both simple MGA trajectories and very complex MGA-1DSM trajectories. PaGMO comes equipped with all the GTOP problems, and this work uses several difficult MGA-1DSM problems from GTOP to serve as test cases (see Chapter V).
IV. IGATO Optimization Engine

This chapter introduces the different elements that make up IGATO's optimization routine. Each section discusses a different strategy that can be applied to global optimization (GO) and the final section combines the individual ideas into one unified algorithm.

IV.I. Introduction

PaGMO has been proven to be highly effective against MGA-DSM problems [17], but it is not perfect and it heavily depends on the quality of the algorithms chosen to inhabit each island (although to a lesser degree than if only using a single algorithm). Therefore, the purpose of this chapter is to investigate different ad-hoc optimization strategies that can be used in conjunction with PaGMO's parallel island model design to improve its performance and reliability. Topics discussed include: restarting the island populations to prevent stagnation, a novel new pruning algorithm which learns over time, subdomain decomposition, and a useful test suite for taking advantage of similarity in good solutions across the search space.

IV.II. Restarts

As a problem is optimized, the diversity of solutions encountered tends to gradually decrease as the optimizer converges to a local optima. Eventually, there may be
practically no diversity at all because the optimizer has fully converged to a particular solution and its local neighborhood. This situation is referred to as stagnation. For this reason, it may be advantageous to run repeated trials of shorter duration rather than one long trial because it promotes better diversity and makes more efficient use of function evaluations. However, shortening each trial in order to run more of them may also be detrimental if taken too far because an optimizer is not given adequate time to converge to a solution. Thus, there is an intrinsic relationship between the number of trials and diversification, and the duration of a trial and intensification. The right balance between the two is often difficult to gauge. One possible way to determine the optimal trial duration would be to run batches of trials at increasing trial durations and try to identify at which point the average solution of a batch is negligibly better than the average solution of the previous batch. This would provide a reasonable estimate that the trial duration associated with the previous batch is "long enough". This experimentally measured optimal trial duration is the nominal evolution time. Running repeated trials for less than or more than the nominal evolution time generally leads to reduced efficiency due to the optimizer favoring diversification over intensification or vice versa.

Determining the nominal evolution time and running repeated trials for that duration can be very efficient, but it still not perfect. Occasionally, an optimizer may converge much sooner than the nominal evolution time suggests in which all function evaluations after that point are essentially wasted. On the flipside, an optimizer may occasionally be prematurely terminated by the nominal evolution time and a potential local optima may be lost. Some problems may have a nominal evolution time with a large standard deviation in which case these occurrences could be regular. An alternative to
using trials, which statically restart an evolution after a fixed stopping criteria, is to use automatic restarts which dynamically restart an evolution once the optimizer thinks that it has fully converged to a solution. An evolution is considered fully converged once it has met the convergence criteria. The convergence criteria used in this work uses the average range of each coordinate among the members of the total population. The minimum and maximum values of each coordinate in the decision vector are calculated from all population members.

\[ x_{i,\text{min}} < x_i \forall x_i \in P \]

\[ x_{i,\text{max}} > x_i \forall x_i \in P \]

The range of each coordinate is the difference between the max and min values:

\[ d_i = x_{i,\text{max}} - x_{i,\text{min}} \]

The average range is then

\[ d_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} d_i \]

The evolution is considered fully converged when \( d_{\text{avg}} \) decreases below the threshold value \( d_{\text{threshold}} \). For this work, a threshold value of 0.1 (\( d_{\text{threshold}} = 0.1 \)) was determined to reflect the optimization convergence fairly well. Other values between 0.05 and 0.5 were tested as well, but proved to either prematurely restart the populations or wait too long. The value of 0.1 works well because its small enough such that there is very little diversity left in the populations and large enough such that the populations don't have to be completely converged, which requires excessive computation time.
IV.III. Interval Based Pruning

Pruning, when applied to global optimization problems, is the act of removing parts of the search space so that the solutions contained within the pruned area no longer have any possibility of being selected. Pruning is useful for decreasing the size of the search space so that an optimizer may explore the remaining parts more thoroughly. However, pruning can be a dangerous endeavor, especially when dealing with highly volatile fitness landscapes like those of MGA-DSM problems, because there is always the unfortunate chance of pruning out the global optima or other desirable solutions. When dealing with extremely multimodal and noisy landscapes, it becomes increasingly difficult to conclude that a particular section of the search space should be pruned or not. As a result, pruning strategies can easily fall victim to accidentally pruning out the optimal solution, or other nearly-optimal solutions, because of the chance of \textit{desirable} solutions hiding within what we believe are \textit{undesirable} parts of the search space. To acquire complete confidence that a section of the search space is truly undesirable would require an exhaustive search which is not the intent of heuristics. Thus, using pruning strategies with heuristics is considered a high-risk high-reward scenario. If an algorithm is fortunate enough not to prune the optimal solution, the heuristic stands a much better chance of eventually discovering it. However if it does prune the global optima, then it has essentially shot itself in the foot for the remainder of the optimization; good solutions could potentially still be found, but the global optima has no chance at all.

Despite their drawbacks, pruning strategies can still be powerful tools for heuristics. They allow a global optimizer to more fully explore desirable parts of the search space using far less function evaluations than would be required otherwise. Thus,
pruning can greatly increase the efficiency of a global optimizer so long as we prevent desirable solutions from being pruned. Now while it may not be possible to guarantee that optimal solutions are not pruned, this section describes a pruning strategy that aims to make the best decisions possible using information it continuously gathers throughout the optimization.

The pruning strategy described in this section is based on the idea of subdividing the bounds around each decision variable into $n$ intervals.

$$[a, b] = [a_1, b_1] \cup [a_2, b_2] \cup \ldots \cup [a_i, b_i] \cup \ldots \cup [a_n, b_n]$$

Initially, all intervals are consecutive and equal in length

$$a_i = b_{i-1} \quad \text{for} \quad i > 1$$

$$|b_1 - a_1| = |b_2 - a_2| = \ldots = |b_i - a_i| = \ldots = |b_n - a_n|$$

For example, consider a function which has two decision variables, $x_1$ and $x_2$ both bounded from -1 to 1. For simplicity let $n=4$. Then the search space is simply a 4x4 grid.

![Search Space Diagram](image)

*Figure 4.1. The search space in two dimensions with four intervals along each dimension. The dotted lines represent the grid which is created.*
Unlike many deterministic optimizers, we are not concerned with the 25 node points that are created by the intersection of the grid lines because the number of nodes increases exponentially with the dimension of the problem. Since MGA-DSM problems typically have 10-26 dimensions or more, tracking the number of nodes becomes unfeasible. Instead, we will focus on the intervals that are created along each dimension whose number only increases linearly with the number of dimensions. Table 4.1 compares the number of intervals with the number of nodes from 2-26 dimensions with 10 intervals per coordinate.

<table>
<thead>
<tr>
<th>Number of dimensions</th>
<th>Number of Intervals</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2*10=20</td>
<td>10^2 = 100</td>
</tr>
<tr>
<td>3</td>
<td>3*10=30</td>
<td>10^3 = 1000</td>
</tr>
<tr>
<td>6</td>
<td>6*10=60</td>
<td>10^6 = 1,000,000</td>
</tr>
<tr>
<td>9</td>
<td>9*10=90</td>
<td>10^9 = 1,000,000,000</td>
</tr>
<tr>
<td>18</td>
<td>18*10=180</td>
<td>10^18</td>
</tr>
<tr>
<td>26</td>
<td>26*10=260</td>
<td>10^{26}</td>
</tr>
</tbody>
</table>

In order to determine which intervals to prune, the intervals must first be ranked by using information gathered from the search space. As the optimization runs, the intervals archive bits of information which help deduce how desirable certain intervals are relative to the others.
IV.III.I. Interval Archiving

Each i'th interval keeps track of the best solution discovered which has its i'th coordinate within the interval boundaries. The global best solution is always one of the best solutions for an interval in every dimension. It's easiest to use an example to illustrate the archiving process. Consider again the MATLAB Peaks function from equation 3.5. Table 4.2 displays the first six made-up solutions that an arbitrary global optimizer found while trying to minimize the Peaks function.

<table>
<thead>
<tr>
<th>Solution Number</th>
<th>Objective function, f</th>
<th>X₁</th>
<th>X₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.7</td>
<td>-0.2</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>0.7</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.4</td>
<td>-0.6</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.5</td>
<td>-0.4</td>
</tr>
<tr>
<td>5</td>
<td>-0.5</td>
<td>-0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>6</td>
<td>1.8</td>
<td>-0.1</td>
<td>-0.8</td>
</tr>
</tbody>
</table>

Each solution gets archived immediately after its objective function is calculated. The first solution found has coordinates $(-0.2, 0.9)$ so we record the cost $f = 2.7$ in the second interval of the first dimension and the fourth interval of the second dimension. We do something similar for the second and third solutions.
The fourth solution's $x_1$ coordinate belongs to an interval which already has a recorded solution. In this situation, the objective function of the stored solution is compared to that of the incoming solution. If the incoming solution is better, as in this case it is, it replaces the existing solution. If the incoming solution is worse than the existing solution, then the incoming solution is rejected. The last two solutions also have conflicts which must be sorted out in a similar manner.

Figure 4.2. Demonstration of the interval archiving procedure for the first three solutions from Table 4.2.
As new solutions are discovered, they are constantly archived to their appropriate intervals if an improvement in those intervals is evident. As the optimization progresses, the intervals become more representative of the search space and the lowest performing intervals can be pruned.

**IV.III.II. The Affine Space**

The problem with the interval archiving method described in 4.3.1 is that pruning any interval other than the first or last one will result in discontinuous search space which violates the continuity requirements on many global optimizers. Limiting the pruning to the edge of the boundaries essentially defeats the point of using intervals altogether, and we would be better off using cluster-pruning strategies. In order to use intervals to their
full extent, one possible solution is to use a transformation which can transform the discontinuous intervals in the *real space* into continuous intervals in the *affine space*. The transformation is termed the *affine transformation* [1]. The transformation offers the additional advantage of being able to easily scale each dimension so that the resulting lower and upper bounds are zero and one respectively. Scaling in this manner results in the search space taking the form of a unit hyperrectangle in the affine space. Each dimension of the problem has its own affine transformation, so in general, there are $n$ affine transformations. An affine transformation is defined by the set of linear parametric equations which map a point $x_{i,j}$ belonging to the $j$'th interval of the $i$'th dimension in the affine space to its corresponding point $\tilde{x}_{i,j}$ belonging to the $j$'th interval of the $i$'th dimension in the real space and vice versa.

\[
x_{i,j} = a_{i,j} + \frac{(b_{i,j} - a_{i,j})}{(\tilde{b}_{i,j} - \tilde{a}_{i,j})}(\tilde{x}_{i,j} - \tilde{a}_{i,j})
\]

\[
\tilde{x}_{i,j} = \tilde{a}_{i,j} + \frac{(\tilde{b}_{i,j} - \tilde{a}_{i,j})}{(b_{i,j} - a_{i,j})}(x_{i,j} - a_{i,j})
\]

\[
0 \leq i < n, \quad 0 \leq j < m
\]

where $a_{i,j}$ and $b_{i,j}$ are the lower and upper bounds of the $j$'th interval belonging to the $i$'th dimension in real space and in the same manner, $\tilde{a}_{i,j}$ and $\tilde{b}_{i,j}$ are the lower and upper bounds of the $j$'th interval belonging to the $i$'th dimension in affine space. Again, a simple example is useful to clarify the point. Here, there is only one dimension $n=1$ and two intervals $m=2$. 

Notice that in the above example, the intervals in real space are different lengths and discontinuous, but their corresponding intervals in the affine space are continuous and equal in length. Half of the affine space maps to the first interval in real space and the other half of the affine space maps to the second interval in real space.

The affine space grants the pruning algorithm the freedom to prune any interval in the real space it chooses while still keeping the optimization box-constrained so that any generic optimizer may be used. However, in order to calculate the objective function, the decision vector in affine space must first be un-transformed to its real-valued counterpart. Therefore, the objective function must wrapped in a similar manner to [1].

\[
\tilde{x}_{1,1} = 0 + \frac{(0.5 - 0)}{(10 - 5)} (x_{1,1} - 5) = 0.1(x_{1,1} - 5)
\]

\[
\tilde{x}_{1,2} = 0.5 + \frac{(1 - 0.5)}{(30 - 20)} (x_{1,2} - 20) = 0.5 + 0.05(x_{1,2} - 20)
\]

Figure 4.4. The affine transformation applied to two discontinuous intervals of different length (left). The resulting intervals (right) are continuous and equally spaced.
The immediate reaction to Figure 4.5 might be to ask what about computation time? It may be trivial to identify which intervals hold a point in the affine space when there are only a 10 intervals, but what happens when there are 1000? Checking each interval systematically to see if a point is between its lower and upper bound is cumbersome and computation time increases linearly with the number of intervals. Luckily, there is an alternative method for identifying containing intervals which has nearly constant time complexity. Regardless of how the real space is sliced and diced during the pruning process, the affine space always stays the same: It is always the unit hyperrectangle with \( m \) intervals along each dimension. What are changing are the linear mappings between

---

**Figure 4.5. The wrapped objective function.** Each decision vector must be transformed back into the real space before the objective function can be calculated.
the affine and real spaces, but not the intervals in the affine space themselves. This means that if \( m=4 \), the affine space will always have the four consecutive intervals:

\[
\tilde{x}_1 \begin{bmatrix} 0 & 1 & 2 & 3 \\ [0, 0.25) & (0.25, 0.5) & (0.5, 0.75) & (0.75, 1) \end{bmatrix}
\]

**Figure 4.6. A coordinate in the affine space with four intervals. Their boundaries are always the same.**

during the entire optimization process. Therefore, if a point in the affine space has a \( \tilde{x}_1 \) coordinate of 0.6 then, since the affine intervals are consecutive and equally spaced, we know that it will belong to the third interval (index 2) without having to systematically check each interval one at a time. More generally, the \( j \)’th containing interval of a point with coordinate \( \tilde{x}_i \) can be calculated analytically:

\[
j'\text{th interval} = \min \left( m - 1, \text{floor} \left( \frac{m (\tilde{x}_i - \overline{b}_i)}{\overline{u}_i - \overline{b}_i} \right) \right) \tag{4.1}
\]

where \( \overline{b}_i \) and \( \overline{u}_i \) are the lower and upper bounds of the \( i \)’th dimension in the affine space respectively. Using equation 4.1, it becomes trivial to identify which interval contains a point in affine space, which is the same index as the interval in real space, regardless of the number of intervals used, i.e. a nearly constant time calculation.

**IV.IV. Subdomain Decomposition**

The pruning strategy described in section IV.III is a powerful tool for filtering out undesirable regions of the search space, but at the cost of possibly losing good solutions as well. Another useful strategy for reducing the size of a search space without the same consequence is to use a *branching* strategy. Branching gets its name from a tree branch
which starts off as a single limb and then splits off, "branches", into smaller limbs which themselves split into smaller limbs on so on. Branching is useful breaking up a large object into many smaller ones which together comprise the whole. Each individual object can then be analyzed more thoroughly, but every object has to be analyzed in order to be completely thorough. Therefore branching reduces the size of the search space at the cost of splitting resources or increasing computation time, whereas pruning reduces the size of the search space at the cost of potentially losing desirable solutions.

This work makes use of a branching strategy in addition to the pruning method described previously. The initial search space is referred to as the domain of the problem. A simple bisection branching strategy is used which decomposes the domain into smaller and smaller subdomains as the optimization progresses [13]. The optimizer always operates within a single subdomain. The benefit of subdomain decomposition is twofold: First, branching guarantees theoretical absolute convergence which means that if the optimization was allowed to run for infinity, the size of each subdomain would shrink to that of a single point in which complete knowledge of the system is achieved. Secondly, subdomains naturally promote better diversity because a given evolution is restricted to finding solutions only within the active subdomain bounds. Therefore if there are strong local attractors outside of a subdomain, an evolution on that subdomain will have no chance of converging to them.
The second benefit is of particular interest to MGA-DSM problems because spacecraft mission analysts often desire a variety of trajectory options for different launch windows. Traditionally subdomain decomposition works best when each bisection is always along one of the longest edges of the subdomain. This implies that no coordinate will be cut \(n+2\) times before every other coordinate is cut at least \(n\) times. This keeps the subdomains decomposing in a symmetric and uniform manner. However, this work intentionally goes a different direction because in our case it is more desirable to bisect a single coordinate (the departure date) multiple times. Repeatedly bisecting the departure date artificially creates more and more launch window boundaries which force the optimizer to look for good solutions using departure dates they might normally avoid. Thus subdomain decomposition serves the dual purpose of breaking down the search

---

**Figure 4.7.** An active subdomain of the entire domain. Performing an evolution within the subdomain restricts the optimizer from finding solutions outside the subdomain boundaries.
space into smaller chunks to improve intensification and also to encourage a better
diversity of good solutions for different launch windows.

IV.V. Similarity Testing

It's often the case that two local optima may be very similar to each other and yet
very distant from each other in the search space. Therefore, searching in the local
neighborhood of one will not necessarily lead to the discovery of the other. A good
example of this situation is with the Messenger Problem from ESA's GTOP Database
(see test case 1 in Chapter V). The best known solution to this problem has an objective
function value of $f = 8.630$ km/s. This solution is devilishly tricky to find, often
requiring hundreds of millions of function evaluations or several hours computation time,
whereas the local optima with objective function value of $f = 8.701$ km/s can typically
be discovered in a matter of minutes using the PaGMO or IGATO optimizers. The
interesting thing about these two solutions is that a majority of their 18 decision variables
are nearly identical and several of those that are different are still in the near vicinity of
each other. However, a couple of their coordinates (including the departure date) are very
different from each other resulting in the two solutions living in two different areas of the
search space. The two solutions are very similar to each other, and yet the former is
considerably more difficult to find than the latter. This often occurs when a good solution
has a very narrow basin of attraction and is buried in an area that appears much less
desirable than the solution itself or if it happens to be next to local optima with a large
basin of attraction that "steals" all the attention. In cases such as this, search heuristics
may have a very difficult time identifying these well-hidden narrow basins of attraction,
pruning strategies may stand a greater chance of accidentally pruning them based on their surrounding area, and subdomain decomposition may take too long to create a subdomain small enough to make much difference to the search heuristic. In response to these shortcomings, this work includes an additional optimization technique which aims at discovering new desirable solutions that are similar to an established solution. This procedure is referred to as Similarity Testing, and is similar in concept to mutation operators in various global optimizers, but in this case, at a macroscopic scale.

Similarity Testing consists of test suite which operates on a given solution in hopes of discovering a better similar solution. The test suite consists of three independent tests: Single Dimension Test (SDT), Multi Dimension Test (MDT), and Gaussian Neighborhood Test (GNT). In order to perform any of the three tests, a local neighborhood must first be created around the given solution. The local neighborhood creates "wiggle-room" for all the decision variables that are not currently being tested. The (normalized) diameter of the local neighborhood in this work is set to 0.1. For example if the given solution has a decision variable with a value of 0.35, then the local neighborhood for that decision variable is $0.35 \pm 0.05$. The first two tests make use of the local neighborhood.

**Single Dimension Test (SDT)**

The first test is the SDT which enumeratively tests each decision variable one at a time. For this test, the first decision variable is chosen and its neighborhood is set equal to the domain boundaries. All other decision variables are limited to their respective local neighborhood as described above. The archipelago is reinitialized with a random
population and the given solution is inserted into the population on each island. The
archipelago is then allowed to evolve for a specified amount of time (in this work, two
migrations are allowed to occur before ending the evolution). After the evolution
completes, if a better solution was found, then it replaces the initial solution and a new
local neighborhood is constructed around each decision variable of the new solution. If
the evolution failed to find a better solution, then the local neighborhood stays the same.
The second decision variable is then chosen and its neighborhood is set to the domain
boundaries whereas all of decision variables (including the first one) are set to their
respective local neighborhoods and the process repeats. This test continues until the last
decision variable is tested. Below is pseudocode for the SDT:

**Single Dimension Test (SDT) Pseudocode**

1. \textit{for } \( i \leftarrow 1: n \)
2. \( N \leftarrow \text{constructLocalNeighborhood}(x) \)
3. \( N(i) \leftarrow [lb(i), \ ub(i)] \)
4. \( A \leftarrow \text{constructArchipelago}(N,x,f) \)
5. \( \text{evolve}(A) \)
6. \( [x_{\text{new}}, \ f_{\text{new}}] \leftarrow \text{getChampion}(A) \)
7. \textit{if } \( f_{\text{new}} < f \text{ then} \)
8. \( f \leftarrow f_{\text{new}}; \ x \leftarrow x_{\text{new}} \)
9. \( i \leftarrow i + 1 \)
10. \textit{return}(x,f) \)
Here \( n \) is the number of dimensions, \( x \) and \( f \) are the decision vector and objective function of the given solution respectively, \( N \) is the local neighborhood for each decision variable, \( lb \) and \( ub \) are the vectors of lower and upper bounds of the domain respectively, and \( A \) is the archipelago.

**Multi Dimension Test (MDT)**

The second test is the MDT and it is similar to the SDT except that now multiple dimensions are tested simultaneously. For each sub-test, a random number of dimensions \( n_{dim} \) between two and the number of dimensions \( n \) is chosen according to a decaying exponential probability density function (PDF) so that there is a large chance of choosing two dimensions and a very small chance of choosing \( n \) dimensions. Once the number of dimensions is known, the \( n_{dim} \) decision variables to be tested are chosen at random and the length of the test is set proportional to \( n_{dim} \). The test then proceeds in the same manner as the SDT. The bounds around all \( n_{dim} \) chosen decision variables are set to the bounds on the domain while the bounds on all other decision variables are restricted to their local neighborhood. An archipelago is constructed and the given solution is inserted into the population of each island. After evolving for the specified evolution time, a new random number of dimensions is chosen and the process repeats itself until the total number of function evaluations is reached. Below is pseudocode for the MDT:
Multi Dimension Test (MDT) Pseudocode

1. $n_{\text{eval}} \leftarrow 0$
2. while $n_{\text{eval}} < n_{\text{EVAL MAX}}$
3. \hspace{1em} $N \leftarrow \text{constructLocalNeighborhood}(x)$
4. \hspace{1em} $n_{\text{dim}} = \text{randExponential}(2, n)$
5. \hspace{1em} for $i \leftarrow 1:n_{\text{dim}}$
6. \hspace{1.8em} $d \leftarrow \text{rand}(1, n)$
7. \hspace{1.8em} $N(d) \leftarrow [lb(d), \ ub(d)]$
8. \hspace{1.2em} $i \leftarrow i + 1$
9. \hspace{1em} $A \leftarrow \text{constructArchipelago}(N, x, f)$
10. \hspace{1em} $\text{evolve}(A)$
11. \hspace{1.2em} $n_{\text{eval}} \leftarrow n_{\text{eval}} + n_{\text{evolution eval}}$
12. \hspace{1.2em} $[x_{\text{new}}, \ f_{\text{new}}] \leftarrow \text{getChampion}(A)$
13. \hspace{1.2em} if $f_{\text{new}} < f$ then
14. \hspace{2.4em} $f \leftarrow f_{\text{new}}; \ x \leftarrow x_{\text{new}}$
15. \hspace{1.2em} return($x, f$)

Here $n_{\text{eval}}$ is the total number of elapsed function evolutions, $n_{\text{EVAL MAX}}$ is the maximum allowed function evaluations, and $n_{\text{dim}}$ is the random number of dimensions chosen between 2 and $n$ from following a decaying exponential probability function.
Gaussian Neighborhood Test (GNT)

The final test is the GNT which is different from the previous two and does not use the previously described local neighborhood. In this test, the bounds around each decision variable is chosen randomly according to a Gaussian distribution (hence the name). An archipelago is constructed and the given solution is inserted into the population on each island just as before. The archipelago is then evolved until the total number of function evaluations is reached. If dynamic restarts are allowed, then this process can occur multiple times. Below is pseudocode for the GNT:

**Gaussian Neighborhood Test (GNT) Pseudocode**

1. \( n_{eval} \leftarrow 0 \)
2. \( \textbf{while } n_{eval} < n_{EVAL_{MAX}} \)
3. \( \textbf{for } i \leftarrow 1 : n \)
4. \( N(i) \leftarrow \text{GaussianDistribution}(x(i)) \)
5. \( i \leftarrow i + 1 \)
6. \( A \leftarrow \text{constructArchipelago}(N, x, f) \)
7. \( \text{evolve}(A) \)
8. \( n_{eval} \leftarrow n_{eval} + n_{evolution_{eval}} \)
9. \( [x_{new}, \ f_{new}] \leftarrow \text{getChampion}(A) \)
10. \( \textbf{if } f_{new} < f \textbf{ then } \)
11. \( f \leftarrow f_{new}; \ x \leftarrow x_{new} \)
12. \( \text{return}(x, f) \)
IV.VI. Optimization Routine Overview

The intent of the IGATO Optimizer Engine is to combine all of the mentioned optimization techniques into one coherent and robust algorithm. The algorithm is driven by IGATO's two fundamental optimization goals:

1. Consistently identify the best known solution to a given trajectory optimization problem using fewer objective functions than PaGMO alone.

2. Identify as many attractive alternative solutions for different launch windows as possible.

The algorithm starts with a subdomain that is initially set equal to the entire domain. As the optimization progresses, this subdomain is divided into smaller subdomains using the Subdomain Decomposition Method (Section IV.IV). For each subdomain, a number of evolutions are performed referred to as the Evolution Loop.

The Evolution Loop

The first evolution of a subdomain is initialized with a random population and evolved until some stopping criteria is met. If the best solution from the evolution $f_{best}$ is better than the best solution within the subdomain $f_{sbest}$, then the subdomain solution is updated. If $f_{best}$ is better than $f_{sbest}$ by at least the threshold value $\Delta f_{threshold}$, then the improvement flag is set to true \textit{isImproved} = \textit{true}. Next, the intervals within the current subdomain are pruned according to the Pruning Algorithm (see Section IV.III). The populations are then re-initialized and if the previous evolution registered an improvement, $x_{sbest}$ is inserted into each population. Re-initializing the populations and inserting $x_{sbest}$ grants the algorithm the opportunity to continue to improve the solution.
by starting with a mostly fresh-slate. If the next evolution leads to further improvement, then $x_{sbest}$ is re-inserted into the following evolution again and so forth. Once $f_{sbest}$ stops sufficiently improving, then $x_{sbest}$ is not re-inserted and the next evolution starts completely fresh. This process continues for $n_{evolutions}$.

- **Dynamic Population Size**
  
  During the evolution loop, there is the option to either use a constant population size for every evolution or to allow it to change over time. For instance, it is often desirable to start off with a large population size for the early evolutions, which encourages diversification, and gradually decrease the population size each evolution to place more emphasis towards intensification.

- **Pruning**
  
  Throughout the evolution loop, solutions are constantly archived to their respective intervals. After each evolution completes, there is the opportunity to prune a number of intervals. A good strategy determined in this work is to wait until half of the evolutions are complete before starting to prune. This allows the early evolutions to "feel out" the search space and helps prevent the pruning algorithm from making a poor decision too early because it hasn't gained enough information. Once half of the evolutions are complete, the pruning algorithm can begin pruning a select number of intervals after each of the remaining evolutions.

**Similarity Testing**

Once the evolution loop is completed for a particular subdomain, the Similarity Testing Suite (Section IV.V) is applied two times. The first time, the best solution within
the current subdomain $x_{sbest}$ is used as the reference solution whereas the second time
the best solution within the total domain $x_{gbest}$ is used as the reference. The purpose of
these tests is to look for solutions that are very similar to $x_{sbest}$ and $x_{gbest}$ but may be
very distant from each other in the search space. This is particularly important for $x_{sbest}$,
because normally it is restricted by the subdomain boundaries during an evolution loop.
Testing $x_{gbest}$ after each evolution loop is geared towards satisfying goal #1 in which
there is a constant effort in refining and improving the best solution. This is particularly
important if a large number of subdomain divisions are allowed in which case the total
time spent analyzing the subdomain that (unknowingly) holds the global optima grows
increasingly small.

**Subdomain Decomposition**

Following the similarity tests, if the number of subdomains is less than the
maximum number allowed $n_{max\_subdomains}$, then the current subdomain is subdivided
into two children using the Subdomain Decomposition Method (section IV.IV) and the
next subdomain to be explored is chosen. Once the number of subdomains has reached
the limit, then subdomains are no longer subdivided and an existing subdomain is chosen
for the next round. Below, Figure 4.8 gives an overview of the optimization routine
which comprises of the elements discussed in this chapter.
Figure 4.9 below briefly portrays how the optimization routine is implemented in code.

The Optimizer class is the primary high-level class which orchestrates the optimization logic. This class creates instances of the CustomProblem, Archipelago, Domain classes respectively. The optimizer class has four primary member functions (methods): Run(), EvolutionLoop(), SimilarityTests(), and Decompose(). It’s important to note that these functions do not necessarily have null arguments in the code. The first method Run(), is primary function which executes the entire optimization routine and the last three functions correspond to the three major elements of the optimization routine (the three blocks on the left side of Figure 4.8). The Interrupt() function is used to stop an ongoing evolution if a dynamic restart is requested and the Terminate() function stops the entire optimization routine.

The CustomProblem class is derived from PaGMO's Problem::base class and allows the creation of user-defined trajectories from the GUI. The CustomProblem class contains an instance of the
The **MGA_DSM** class which is the implementation of the trajectory model used in this work (see Chapter II). Calling the method **ObjFun()** results in calling the MGA_DSM's **Execute()** method which evaluates the objective function for a given state vector.

The **Archipelago** class is a PaGMO class which creates and handles the asynchronous island model. It has one primary method **Evolve()** which causes each island to perform an evolution in parallel.

The **Domain** class manages the subdomain decomposition, affine transformations, and pruning, and holds a container of **Subdomain** objects (only one initially). The domain class has four primary methods: **UndoAffineTransformation()**, **Archive()**, **PruneSubdomain()**, and **BisectSubdomain()**. The first function is called immediately before the objective function is evaluated for a given state vector in order to transform it back into real space. The domain simply identifies which subdomain holds the state vector in question and calls that subdomain's **UndoAffineTransformation()** function. The **Archive()** function is called immediately following the evaluation of the objective function and again identifies which subdomain holds the state vector (this information is stored between **UndoAffineTransformation()** and **Archive()** calls in order to boost performance) and calls that subdomain's **Archive()** function. The functions **PruneSubdomain()** and **BisectSubdomain()** call the active subdomain's **Prune()** and **Bisect()** methods respectively. The **Subdomain** object has four primary methods which are analogous to four primary methods of the domain class. The first function **UndoAffineTransformation()**, transforms a given state vector into real space, **Archive()** is called to archive the relevant information about a given state vector to each of its corresponding intervals. The function **Prune()** prunes a select number of intervals from each dimension, and **Bisect()** splits the domain into two halves.
<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Domain</th>
<th>Subdomain(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitArchipelago()</td>
<td>UndoAffineTransformation()</td>
<td>UndoAffineTransformation()</td>
</tr>
<tr>
<td>InitProblem()</td>
<td>Archive()</td>
<td>Archive()</td>
</tr>
<tr>
<td>InitDomain()</td>
<td>PruneSubdomain()</td>
<td>Prune()</td>
</tr>
<tr>
<td>Run()</td>
<td>BisectSubdomain()</td>
<td>Bisect()</td>
</tr>
<tr>
<td>EvolutionLoop()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimilarityTests()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decompose()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminate()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Archipelago Evolve()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CustomProblem ObjFun()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MGA_DSM Execute()</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.9. An overview of the Optimizer class and its children in IGATO along with some of their major methods.
V. Test Cases & Validation

This chapter examines multiple test cases and compares the results to PaGMO and other solutions in the GTOP database for comparison and validation. The first section describes the general setup and configuration of the optimizer. Next, three challenging MGA-1DSM problems are then analyzed from the GTOP database. For each problem, a detailed comparison is made between the performance of IGATO and PaGMO. Lastly, a fourth problem involving a round-trip scenario and multiple DSMs on a single planet-to-planet leg is presented which seeks to highlight IGATO's versatile trajectory model implementation.

V.I. Setup

This section describes the general archipelago and optimization settings used in all the following test cases. The archipelago consists of five islands in a "Rim" topology in which four islands make up an outer ring with the additional island in the center. This topology has proven to be effective [17] on MGA-DSM trajectories. A "dream team" of algorithms was chosen to inhabit the islands of the archipelago. To determine which algorithms to select into the dream team, a large pool of different algorithms along with different variants of the same algorithms was constructed. Each algorithm within the pool was tested on several MGA-DSM trajectories by itself and its results were compared to its peers. At the end, the top performers were determined to be Particle Swarm Optimization (variant PSO52), two different variants of Differential Evolution (DE2 and
DE5), and the Corana variant of Adaptive Simulated Annealing (SA Corana). Since Differential Evolution and Adaptive Simulated Annealing were clearly the strongest two performers they were both elected to inhabit two islands each. Particle Swarm Optimization would occupy the remaining island. In the Rim topology, the central island servers an important role. It receives the best solutions from all four of the other islands through migration, optimizes it and sends it back to the outer ring. Typically, this island would be a good home for a local optimizer such as one from the open-source NLopt library [17]. PaGMO provides a wrapper for NLopt among other similar libraries, but unfortunately the author could not get them working correctly. Therefore as a replacement, Adaptive Simulated Annealing was chosen to occupy the central island. Although not a true local optimizer, SA Corana still proved effective at refining solutions migrating from other islands. Therefore the topology was configured with SA Corana on the central island and, in clockwise order, DE2, PSO52, DE5, and another SA Corana occupying the outer ring. The selection policy was set to "Best" which means at the end of an evolution each island selects its best solution and places it into the island's outbox to be sent to all of its connected neighbors. The replacement policy on all islands was set to "Fair" which means before an evolution begins, an island checks its inbox for any solutions arriving from other islands and replaces the worst possible solution in its population with the inbound solution if an improvement is possible. The figure below summarizes the archipelago setup (the "S" stands for selection and "R" stands for replacement):
Since the aim of this work was to test the high level optimization techniques discussed in Chapter IV, and not fine tune each algorithm, the parameters used for each algorithm were generally left to their defaults. The algorithms were configured as shown below.

Table 5.1. Parameter configuration of the Adaptive Simulated Annealing algorithm (SA Corana) in PaGMO.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Temperature</td>
<td>1</td>
</tr>
<tr>
<td>Final Temperature</td>
<td>0.1</td>
</tr>
<tr>
<td>Step Adjustment</td>
<td>1</td>
</tr>
<tr>
<td>Bin Size</td>
<td>20</td>
</tr>
<tr>
<td>Range</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.1. The "Rim" island topology used for all test cases for IGATO and PaGMO along with the algorithm and migration policy assigned to each island.
This concludes the archipelago settings. This archipelago was applied to all four test cases and was also the archipelago used by PaGMO for all performance comparisons. The optimizer settings were typically problem dependent as some problems were much more difficult (requiring more function evaluations) than others. The objective function for all test cases was the one presented by Equation 3.2 which is the summation of the
launch $\Delta V$ plus all DSM $\Delta V$'s plus the difference in velocity between the spacecraft and the arrival planet.

V.II. GTOP Test Cases

The following three cases were taken from the GTOP database. These problems range from 18-26 variables and are some of the most difficult problems in the database. Testing IGATO's performance on each problem against PaGMO proved difficult due to the different nature of the optimizers. The premature convergence inherent in many metaheuristics and PaGMO's lack of capability to recover from such convergence prevents PaGMO from being run indefinitely on a given problem. Instead, PaGMO proves extremely effective when performing repeated trials of much shorter length. Therefore common statistics for PaGMO may include the average and standard deviation fitness of the best solution of each trial. IGATO, On the other hand, was designed to be run once and to discover better and better solutions the longer it is allowed to run. In order to compare the two optimizer's effectiveness against each other the following strategy was used: PaGMO was run 1000 times on each of the following problems with each trial ranging from a minimum to a maximum number of function evaluations (NFE) depending on the problem. Using this data, observations could be made on the likelihood of PaGMO discovering a particular solution during a single trial. For example, if PaGMO found 10 solutions less than 9 km/s out of 1000 trials, then the probability of getting a solution less than 9 km/s on any particular trial is roughly 1%. If the average number of function evaluations and the average time to complete one trial was 1 million and 12 seconds respectively, then it can be estimated that it takes PaGMO roughly 100 million
function evaluations or 20 minutes to find a solution less than 9 km/s for that problem. These types of statistics prove to be much easier for IGATO to compare against. To highlight the benefit of the subdomain decomposition technique, the performance of IGATO and PaGMO is compared after each time IGATO completes a level. A level indicates how many times the subdomain has been bisected. Initially, there is only one subdomain (the domain). The first level is completed after performing the Evolution Loop and Similarity Tests on this subdomain. The second level is completed after splitting the subdomain into two children subdomains (using the subdomain decomposition technique) and performing the Evolution Loop and Similarity Tests on each of them accordingly, and so on. Figure 5.2 below visualizes the concept of levels.

Figure 5.2. The levels of the subdomain decomposition algorithm. Each box represents a subdomain and after each level, each subdomain splits into two children subdomains.
V.II.I. Case 1: Messenger

The first problem analyzed from the GTOP database was the "Messenger" problem. This problem gets its name due to the fact that the planetary sequence Earth-Earth-Venus-Venus-Mercury (EEVVM) is the same as the beginning of the historic Messenger mission. The real Messenger spacecraft also made use of multiple resonant flybys at Mercury before inserting into a planetocentric orbit. There is a more difficult version of this problem in the GTOP database named "Messenger Full" which includes these resonant flybys. This more difficult problem is presented later as the third and final test case. This test case examines the easier version of Messenger which has the state vector and bounds shown below:

Table 5.5. The state vector bounds for the Messenger problem from the GTOP database.

<table>
<thead>
<tr>
<th>State</th>
<th>Variable</th>
<th>LB</th>
<th>UB</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1)</td>
<td>$t_0$</td>
<td>1000</td>
<td>4000</td>
<td>MJD2000</td>
</tr>
<tr>
<td>x(2)</td>
<td>$V_\infty$</td>
<td>1</td>
<td>5</td>
<td>km/sec</td>
</tr>
<tr>
<td>x(3)</td>
<td>u</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(4)</td>
<td>v</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(5)</td>
<td>$T_1$</td>
<td>200</td>
<td>400</td>
<td>days</td>
</tr>
<tr>
<td>x(6)</td>
<td>$T_2$</td>
<td>30</td>
<td>400</td>
<td>days</td>
</tr>
<tr>
<td>x(7)</td>
<td>$T_3$</td>
<td>30</td>
<td>400</td>
<td>days</td>
</tr>
<tr>
<td>x(8)</td>
<td>$T_4$</td>
<td>30</td>
<td>400</td>
<td>days</td>
</tr>
<tr>
<td>x(9)</td>
<td>$\eta_1$</td>
<td>0.01</td>
<td>0.99</td>
<td>days</td>
</tr>
<tr>
<td>x(10)</td>
<td>$\eta_2$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(11)</td>
<td>$\eta_3$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(12)</td>
<td>$\eta_4$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(13)</td>
<td>$R_{p1}$</td>
<td>1.1</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(14)</td>
<td>$R_{p2}$</td>
<td>1.1</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(15)</td>
<td>$R_{p3}$</td>
<td>1.1</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(16)</td>
<td>$b_{m,cl1}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(17)</td>
<td>$b_{m,cl2}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(18)</td>
<td>$b_{m,cl3}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
</tbody>
</table>
The Messenger problem is particularly interesting problem to optimize because there are two other very strong local optima that have an objective function slightly higher than the best known solution. These three local optima have objective functions of 8.701, 8.650, and 8.630 km/s respectively. The solution with objective function 8.650 km/s is very similar to the best known solution, they both reside in the same local neighborhood, and yet it is noticeably easier to find the 8.650 km/s solution than it is to find the best known solution. The solution at 8.701 km/s is much easier to find than the previous two and interestingly enough is very distant from the other two on the fitness landscape. These three solutions are listed in the table below.

Table 5.6. The three best known solutions to the Messenger problem from the GTOP database along with their corresponding state vectors.

<table>
<thead>
<tr>
<th>State</th>
<th>Variable</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1)</td>
<td>$t_0$</td>
<td>1171.645032</td>
<td>1170.42</td>
<td>2369.89</td>
</tr>
<tr>
<td>x(2)</td>
<td>$V_{\infty}$</td>
<td>1.408994</td>
<td>1.4241</td>
<td>1.67208</td>
</tr>
<tr>
<td>x(3)</td>
<td>U</td>
<td>0.379926</td>
<td>0.385884</td>
<td>0.380256</td>
</tr>
<tr>
<td>x(4)</td>
<td>V</td>
<td>0.49800</td>
<td>0.501059</td>
<td>0.499911</td>
</tr>
<tr>
<td>x(5)</td>
<td>$T_1$</td>
<td>400</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>x(6)</td>
<td>$T_2$</td>
<td>178.37225</td>
<td>179.584</td>
<td>168.06</td>
</tr>
<tr>
<td>x(7)</td>
<td>$T_3$</td>
<td>299.2231</td>
<td>299.823</td>
<td>224.695</td>
</tr>
<tr>
<td>x(8)</td>
<td>$T_4$</td>
<td>180.5107</td>
<td>179.972</td>
<td>212.292</td>
</tr>
<tr>
<td>x(9)</td>
<td>$\eta_1$</td>
<td>0.2345946</td>
<td>0.236453</td>
<td>0.237501</td>
</tr>
<tr>
<td>x(10)</td>
<td>$\eta_2$</td>
<td>0.0964769</td>
<td>0.14166</td>
<td>0.0223169</td>
</tr>
<tr>
<td>x(11)</td>
<td>$\eta_3$</td>
<td>0.8299487</td>
<td>0.471123</td>
<td>0.161132</td>
</tr>
<tr>
<td>x(12)</td>
<td>$\eta_4$</td>
<td>0.3171747</td>
<td>0.30753</td>
<td>0.468419</td>
</tr>
<tr>
<td>x(13)</td>
<td>$R_{p1}$</td>
<td>1.80629</td>
<td>1.67603</td>
<td>1.80818</td>
</tr>
<tr>
<td>x(14)</td>
<td>$R_{p2}$</td>
<td>3.041298</td>
<td>2.85634</td>
<td>1.64195</td>
</tr>
<tr>
<td>x(15)</td>
<td>$R_{p3}$</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>x(16)</td>
<td>$b_{\text{int1}}$</td>
<td>1.35077</td>
<td>1.35246</td>
<td>1.29702</td>
</tr>
<tr>
<td>x(17)</td>
<td>$b_{\text{int2}}$</td>
<td>1.09554</td>
<td>1.11861</td>
<td>2.80363</td>
</tr>
<tr>
<td>x(18)</td>
<td>$b_{\text{int3}}$</td>
<td>1.34317</td>
<td>1.34277</td>
<td>1.57266</td>
</tr>
</tbody>
</table>
The goal of this test is to compare the performance differences between IGATO and PaGMO on a difficult MGA-DSM trajectory with several choke points. PaGMO was configured to run 1000 independent repeated trials where each trial ran for a random number of function evaluations between two and five million. Thus, the average number of function evaluations for each PaGMO trial was around 3.5 million.

IGATO was set to run until four complete levels were complete (i.e. 15 subdomain decompositions). This means that the last level consisted of eight subdomains. For each subdomain, the evolution loop consisted of 20 evolutions. Each evolution started with a randomly generated population and could run for a maximum of 10 million function evaluations, but was allowed to dynamically restart sooner if desired. A dynamic restart occurred whenever the average range of all normalized decision variables dropped below the threshold $d_{threshold} = 0.1$. The population size was allowed to decrease linearly as evolutions were completed: The first evolution began with a population size of 50 and the last evolution uses a population size of 20. This encourages more diversification in the earlier stages of the subdomain exploration and greater intensification in the later stages.

The pruning algorithm uses 100 intervals for each decision variable and is constantly archiving data, but it doesn't begin pruning until after the first 10 evolutions. The pruning algorithm prunes a total of 75% of each decision variable over the course of the last 10 evolutions (10 intervals per evolution). Starting the pruning algorithm after half of the evolutions have elapsed helps prevent prematurely pruning attractive areas of the search space by allowing the optimizer apt amount of time to explore the subdomain before the aggressive pruning begins. After each subdomain was explored, two rounds of similarity testing were conducted: the first round on the best solution from the current subdomain
(sbest) and the second round on the best solution in the total domain (gbest). The table below summarizes the IGATO settings:

Table 5.7. The IGATO optimization settings for the Messenger and Cassini2 test cases.

<table>
<thead>
<tr>
<th>IGATO Optimizer Settings: Messenger/Cassini2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels completed:</td>
</tr>
<tr>
<td>4 (8 subdomains in the final level)</td>
</tr>
<tr>
<td>Number of evolutions per subdomain:</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>Max NFE per evolution:</td>
</tr>
<tr>
<td>10 million</td>
</tr>
<tr>
<td>Dynamic restart threshold:</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>Starting population size:</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>Ending population size:</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>Number of intervals per decision variable:</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>Total % of each decision variable pruned:</td>
</tr>
<tr>
<td>75% starting after 10 evolutions</td>
</tr>
<tr>
<td>Similarity Tests:</td>
</tr>
<tr>
<td>sbest &amp; gbest (after every subdomain)</td>
</tr>
</tbody>
</table>

IGATO was run a total of 10 times and the results were averaged. The tables below overview the performance of each optimizer by themselves (Tables 5.8-5.9) and compares them against each other (Table 5.10). Over the course of the 1000 trials, PaGMO was able to find a solution near or less than 8.701 km/s 39 times (3.9%), a solution near or less than 8.650 km/s 13 times (1.3%) and a solution near the best known solution only 3 times (0.3%). Since each trial averaged 3.5 million function evaluations, the estimated number of function evaluations required to find each of the three local optima are listed (along with the estimated run time). The results for IGATO are broken down by level. The best solution, total run time, and total number of function evaluations (averaged across all trials) are shown after completing each level. The third table compares the performance of the two optimizers. On the left, the table reports the
success rate IGATO had in finding each of the three local optima after each level. For example, IGATO was able to find a solution at least as good as 8.701 km/s after the first level every time (100%). On the right side of the table shows PaGMO's estimated success rates which were calculated as:

$$PaGMO_{success\ chance} = 1 - (1 - estimated\ chance\ per\ trial)^{\frac{NFE_{IGATO}}{3.5\ million}}$$

For example, IGATO required 98 million function evaluations in order to complete the first level ($NFE_{IGATO} = 98$ million) which is equivalent to running PaGMO for roughly 28 trials at 3.5 million NFE each. From Table 5.8 we know PaGMO's chance of getting a solution at least as good as 8.701 km/s is 3.9% per trial. So plugging in these numbers yields a 67.2% chance of finding a solution at least as good as 8.701 km/s after the same number of function evaluations as IGATO after the first level. The rest of the values were calculated in a similar manner. The results between the two optimizers are very close on this problem. IGATO is able to find a solution at least as good as 8.701 km/s every time, even after only 1 level, but PaGMO consistently has a slightly higher chance of discovering the best known solution of 8.650 km/s.

<table>
<thead>
<tr>
<th>Benchmark Objective Function</th>
<th>Times found out of 1000 trials</th>
<th>Estimated percent chance per trial</th>
<th>Estimated NFE required (million)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq 8.701$ km/s</td>
<td>39</td>
<td>3.9%</td>
<td>86.06</td>
</tr>
<tr>
<td>$\leq 8.650$ km/s</td>
<td>13</td>
<td>1.3%</td>
<td>258.18</td>
</tr>
<tr>
<td>$\approx 8.630$ km/s</td>
<td>3</td>
<td>0.3%</td>
<td>1118.77</td>
</tr>
</tbody>
</table>
Table 5.9. IGATO results for the GTOP Messenger problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of Subdomains Searched</th>
<th>Average NFE required (million)</th>
<th>Average Best Solution (km/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This Level</td>
<td>Total</td>
<td>98</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>315</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>771</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>15</td>
<td>1675</td>
</tr>
</tbody>
</table>

Table 5.10. Comparison between IGATO and PaGMO's success rates on the GTOP Messenger problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>NFE (mil)</th>
<th>IGATO Success Rate</th>
<th>PaGMO Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Equivalent #Trials</td>
<td></td>
</tr>
<tr>
<td></td>
<td>≤ 8.701</td>
<td>≤ 8.650</td>
<td>≈ 8.630</td>
</tr>
<tr>
<td>1</td>
<td>98</td>
<td>100%</td>
<td>67.2%</td>
</tr>
<tr>
<td>2</td>
<td>315</td>
<td>100%</td>
<td>97.2%</td>
</tr>
<tr>
<td>3</td>
<td>771</td>
<td>100%</td>
<td>99.9%</td>
</tr>
<tr>
<td>4</td>
<td>1675</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 5.3 below shows the resulting trajectory plot of the best solution found. The departure and arrival locations are labeled in gray. The white hollow circles are the locations of the flyby events and the white stars are the locations of the DSMs. Figure 5.4 below shows a Porkchop plot (see the user guide Chapter VI) for the Messenger problem generated by IGATO. The location of the planets are shown at the end of the mission. For a complete trajectory breakdown, see Appendix A.
Figure 5.3. IGATO trajectory plot of the best found solution (f=8.630 km/s) to the GTOP Messenger problem.

Figure 5.4. IGATO Porkchop plot for the GTOP Messenger problem.
V.II.II. Case 2: Cassini 2

The second problem investigated from the GTOP database was Cassini2. This trajectory is again inspired by a historic mission: the Cassini probe. The planetary sequence is Earth-Venus-Venus-Earth-Jupiter-Saturn (EVVEJS) with one DSM between each planet which results in the 22 variable decision vector shown below:

<table>
<thead>
<tr>
<th>State</th>
<th>Variable</th>
<th>LB</th>
<th>UB</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1)</td>
<td>$t_0$</td>
<td>-1000</td>
<td>0</td>
<td>MJD2000</td>
</tr>
<tr>
<td>x(2)</td>
<td>$V_0$</td>
<td>3</td>
<td>5</td>
<td>km/sec</td>
</tr>
<tr>
<td>x(3)</td>
<td>$u$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(4)</td>
<td>$v$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(5)</td>
<td>$T_1$</td>
<td>100</td>
<td>400</td>
<td>days</td>
</tr>
<tr>
<td>x(6)</td>
<td>$T_2$</td>
<td>100</td>
<td>500</td>
<td>days</td>
</tr>
<tr>
<td>x(7)</td>
<td>$T_3$</td>
<td>30</td>
<td>300</td>
<td>days</td>
</tr>
<tr>
<td>x(8)</td>
<td>$T_4$</td>
<td>400</td>
<td>1600</td>
<td>days</td>
</tr>
<tr>
<td>x(9)</td>
<td>$T_5$</td>
<td>800</td>
<td>2200</td>
<td>days</td>
</tr>
<tr>
<td>x(10)</td>
<td>$\eta_1$</td>
<td>0.01</td>
<td>0.9</td>
<td>n/a</td>
</tr>
<tr>
<td>x(11)</td>
<td>$\eta_2$</td>
<td>0.01</td>
<td>0.9</td>
<td>n/a</td>
</tr>
<tr>
<td>x(12)</td>
<td>$\eta_3$</td>
<td>0.01</td>
<td>0.9</td>
<td>n/a</td>
</tr>
<tr>
<td>x(13)</td>
<td>$\eta_4$</td>
<td>0.01</td>
<td>0.9</td>
<td>n/a</td>
</tr>
<tr>
<td>x(14)</td>
<td>$\eta_5$</td>
<td>0.01</td>
<td>0.9</td>
<td>n/a</td>
</tr>
<tr>
<td>x(15)</td>
<td>$R_{p1}$</td>
<td>1.05</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(16)</td>
<td>$R_{p2}$</td>
<td>1.05</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(17)</td>
<td>$R_{p3}$</td>
<td>1.15</td>
<td>6.5</td>
<td>n/a</td>
</tr>
<tr>
<td>x(18)</td>
<td>$R_{p4}$</td>
<td>1.7</td>
<td>291</td>
<td>n/a</td>
</tr>
<tr>
<td>x(19)</td>
<td>$b_{mcl1}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(20)</td>
<td>$b_{mcl2}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(21)</td>
<td>$b_{mcl3}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(22)</td>
<td>$b_{mcl4}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
</tbody>
</table>

Even though Cassini2 has four more variables than Messenger, their difficulty is fairly similar. Cassini2 doesn't have a few dominant local optima like Messenger did, there are
many local optima between roughly 8.75 km/s and 8.383 km/s, so the results were
compiled slightly differently. Instead of looking for the success rate of getting into the
neighborhood of local optima as with Messenger, this test looks for solutions below the
following thresholds: 8.55, 8.45, 8.40, and 8.39 km/s. The best known solution from the
GTOP database has an objective function of 8.383 km/s and its state vector is shown
below.

Table 5.12. The best known solution to the Cassini2 problem from the GTOP database.

<table>
<thead>
<tr>
<th>Cassini 2: Best Known Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
</tr>
<tr>
<td>f</td>
</tr>
<tr>
<td>x(1)</td>
</tr>
<tr>
<td>x(2)</td>
</tr>
<tr>
<td>x(3)</td>
</tr>
<tr>
<td>x(4)</td>
</tr>
<tr>
<td>x(5)</td>
</tr>
<tr>
<td>x(6)</td>
</tr>
<tr>
<td>x(7)</td>
</tr>
<tr>
<td>x(8)</td>
</tr>
<tr>
<td>x(9)</td>
</tr>
<tr>
<td>x(10)</td>
</tr>
<tr>
<td>x(11)</td>
</tr>
<tr>
<td>x(12)</td>
</tr>
<tr>
<td>x(13)</td>
</tr>
<tr>
<td>x(14)</td>
</tr>
<tr>
<td>x(15)</td>
</tr>
<tr>
<td>x(16)</td>
</tr>
<tr>
<td>x(17)</td>
</tr>
<tr>
<td>x(18)</td>
</tr>
<tr>
<td>x(19)</td>
</tr>
<tr>
<td>x(20)</td>
</tr>
<tr>
<td>x(21)</td>
</tr>
<tr>
<td>x(22)</td>
</tr>
</tbody>
</table>
The tests were then proceeded in a similar manner to Messenger. PaGMO was run for 1000 trials with each trial running for a random number of function evaluations between two and five million. The tables below overview the performance for each optimizer individually and then compares them against each other. The advantage of IGATO in this test case is much more pronounced. IGATO could successively find a solution close to the best known solution (< 8.39 km/s) 60% of the time after only the first level and 100% of the time after the second level. PaGMO on the other hand, has a very difficult time discovering the best known solution: discovering only 1 solution out of a 1000 that was less than 8.39 km/s. IGATO's dominance on this problem is not surprising, most of the good solutions are located in one region of the search space which makes IGATO's subdomain decomposition, pruning techniques, and similarity testing particularly effective.

Table 5.13. PaGMO results for GTOP Cassini2 problem.

<table>
<thead>
<tr>
<th>Benchmark Objective Function</th>
<th>Times found out of 1000 trials</th>
<th>Estimated percent chance per trial</th>
<th>Estimated NFE required (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 8.55 km/s</td>
<td>48</td>
<td>4.8%</td>
<td>69.08</td>
</tr>
<tr>
<td>&lt; 8.45 km/s</td>
<td>16</td>
<td>1.6%</td>
<td>207.08</td>
</tr>
<tr>
<td>&lt; 8.4 km/s</td>
<td>4</td>
<td>0.4%</td>
<td>828.91</td>
</tr>
<tr>
<td>&lt; 8.39 km/s</td>
<td>1</td>
<td>0.1%</td>
<td>3315.65</td>
</tr>
</tbody>
</table>
Table 5.14. IGATO results for the GTOP Cassini2 problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of Subdomains Completed</th>
<th>Average NFE required (millions)</th>
<th>Average Best Solution (km/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This Level</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>217</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>602</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>1384</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>15</td>
<td>2923</td>
</tr>
</tbody>
</table>

Table 5.15. Comparison between IGATO and PaGMO's success rates on the GTOP Cassini2 problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>NFE (mil)</th>
<th>IGATO Success Rate</th>
<th>PaGMO Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>≤ 8.55  ≤ 8.45 ≤ 8.40 ≤ 8.39</td>
<td>≤ 8.55  ≤ 8.45 ≤ 8.40 ≤ 8.39</td>
</tr>
<tr>
<td>1</td>
<td>217</td>
<td>90% 90% 70% 60%</td>
<td>62 95.3% 63.2% 22.0% 6.0%</td>
</tr>
<tr>
<td>2</td>
<td>602</td>
<td>100% 100% 100% 100%</td>
<td>172 100% 93.8% 49.8% 15.8%</td>
</tr>
<tr>
<td>3</td>
<td>1384</td>
<td>100% 100% 100% 100%</td>
<td>395.4 100% 99.8% 79.5% 32.7%</td>
</tr>
<tr>
<td>4</td>
<td>2923</td>
<td>100% 100% 100% 100%</td>
<td>835.1 100% 100% 96.5% 56.6%</td>
</tr>
</tbody>
</table>

Figures 5.5 shows the resulting trajectory plot of the best solution found. In addition, figure 5.6 shows the IGATO generated Porkchop plot for the Cassini2 problem. For a complete trajectory breakdown, see Appendix A.
Figure 5.5. IGATO trajectory plot for the best found solution (v=8.3830 km/s) to the Cassini2 GTOP problem.

Figure 5.6. IGATO Porkchop plot for the Cassini2 GTOP problem.
V.II.III. Case 3: Messenger Full

The last test case examined is the full blown Messenger Full problem. This is the most difficult problem in the GTOP database, partly because it has 26 variables, but mostly because the resonant flybys added at the end creates an extremely chaotic and challenging search space. The bounds on the decision vector are shown below.

<table>
<thead>
<tr>
<th>State</th>
<th>Variable</th>
<th>LB</th>
<th>UB</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1)</td>
<td>$t_0$</td>
<td>1900</td>
<td>2300</td>
<td>MJD2000</td>
</tr>
<tr>
<td>x(2)</td>
<td>$V_\infty$</td>
<td>2.5</td>
<td>4.05</td>
<td>km/sec</td>
</tr>
<tr>
<td>x(3)</td>
<td>$u$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(4)</td>
<td>$v$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(5)</td>
<td>$T_1$</td>
<td>100</td>
<td>500</td>
<td>days</td>
</tr>
<tr>
<td>x(6)</td>
<td>$T_2$</td>
<td>100</td>
<td>500</td>
<td>days</td>
</tr>
<tr>
<td>x(7)</td>
<td>$T_3$</td>
<td>100</td>
<td>500</td>
<td>days</td>
</tr>
<tr>
<td>x(8)</td>
<td>$T_4$</td>
<td>100</td>
<td>500</td>
<td>days</td>
</tr>
<tr>
<td>x(9)</td>
<td>$T_5$</td>
<td>100</td>
<td>500</td>
<td>days</td>
</tr>
<tr>
<td>x(10)</td>
<td>$T_6$</td>
<td>100</td>
<td>600</td>
<td>days</td>
</tr>
<tr>
<td>x(11)</td>
<td>$\eta_1$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(12)</td>
<td>$\eta_2$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(13)</td>
<td>$\eta_3$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(14)</td>
<td>$\eta_4$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(15)</td>
<td>$\eta_5$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(16)</td>
<td>$\eta_6$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(17)</td>
<td>$R_{p1}$</td>
<td>1.1</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(18)</td>
<td>$R_{p2}$</td>
<td>1.1</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(19)</td>
<td>$R_{p3}$</td>
<td>1.05</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(20)</td>
<td>$R_{p4}$</td>
<td>1.05</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(21)</td>
<td>$R_{p5}$</td>
<td>1.05</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(22)</td>
<td>$b_{mcl1}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(23)</td>
<td>$b_{mcl2}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(24)</td>
<td>$b_{mcl3}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(25)</td>
<td>$b_{mcl4}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(26)</td>
<td>$b_{mcl5}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
</tbody>
</table>
Unfortunately, the decision vector of the best known solutions to this problem are not available in the GTOP database, but the best objective function at the time of writing is reported to be 2.970 km/s. Keep in mind, however, that the 2nd best solution is reported to be 4.254 km/s, the third best is 6.047 km/s, and the fifth best is 6.943 km/s. This is a very wide range of solutions which goes to show how difficult and unpredictable the search space is. So far to date, IGATO has yet to find a solution below the 7 km/s barrier and PaGMO alone fails to find a solution less than 8 km/s after over 2000 trials with each trial running for a random number of function evaluations between 10 and 20 million NFE. These results are disappointing to say the least. One possible theory why IGATO fails to find better caliber solutions is because the subdomains are always bisected along the first coordinate (the departure date). This strategy was chosen over other logical alternatives (such as always cutting along a largest edge) because this work places emphasis on identifying as many good solutions in different launch windows possible. Continuously bisecting the departure date directly enforces an increasing number of launch windows the longer the optimizer is ran. The reason this strategy may not be well suited for Messenger is because Messenger already has fairly tight bounds around its departure date to begin with. Whereas the previous problems had a range of 1000's of days, Messenger's departure date has a range of only 400 days. After a few bisections, the range of departure dates for each subdomain are already so small that future bisections offer little benefit. One possible solution to this problem is to enforce a minimum range on each coordinate. If the difference between the departure date bounds becomes less than the minimum range, then future bisections choose a new coordinate to cut along, ideally the new coordinate is another variable of time (such as T1, T2, etc.).
Unfortunately, this strategy was not implemented due to lack of time. Regardless of IGATO and PaGMO's poor performance in respect to the best solutions in the database, the results are still shown below. Similar to Cassini2, four thresholds were established at 11 km/s, 10 km/s, 9 km/s and 8 km/s and comparisons between IGATO and PaGMO were carried out in a similar manner to the previous two cases. As mentioned before, each trial run by PaGMO was allowed to evolve for a random NFE between 10 and 20 million, a sizable increase over the earlier missions. In addition IGATO proceeded one level deeper, ending with 16 subdomains on the final level, and the max NFE for each evolution was increased from 10 million to 25 million. The rest of the settings are identical to Table 5.7 and they are reprinted again below.

<table>
<thead>
<tr>
<th>IGATO Optimizer Settings: Messenger Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels completed:</td>
</tr>
<tr>
<td>(16 subdomains in the final level)</td>
</tr>
<tr>
<td>Number of evolutions per subdomain:</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>Max NFE per evolution:</td>
</tr>
<tr>
<td>25 million</td>
</tr>
<tr>
<td>Dynamic restart threshold:</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>Starting population size:</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>Ending population size:</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>Number of intervals per decision variable:</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>Total % of each decision variable pruned:</td>
</tr>
<tr>
<td>75% starting after 10 evolutions</td>
</tr>
<tr>
<td>Similarity Tests:</td>
</tr>
<tr>
<td>sbest &amp; gbest (after every subdomain)</td>
</tr>
</tbody>
</table>

Below are the tabulated results summarizing IGATO and PaGMO's performance individually and against each other.
Table 5.18. PaGMO results for GTOP Messenger Full problem.

<table>
<thead>
<tr>
<th>Benchmark Objective Function</th>
<th>Times found out of 1000 trials</th>
<th>Estimated percent chance per trial</th>
<th>Estimated NFE required (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 11 km/s</td>
<td>31</td>
<td>3.1%</td>
<td>477</td>
</tr>
<tr>
<td>&lt; 10 km/s</td>
<td>13</td>
<td>1.3%</td>
<td>1138</td>
</tr>
<tr>
<td>&lt; 9 km/s</td>
<td>5</td>
<td>0.05%</td>
<td>2960</td>
</tr>
<tr>
<td>&lt; 8 km/s</td>
<td>0</td>
<td>0%</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 5.19. IGATO results for the GTOP Messenger Full problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of Subdomains</th>
<th>Average NFE required</th>
<th>Average Best Solution (km/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This Level</td>
<td>Cumulative</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>383</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>922</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>15</td>
<td>2011</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>31</td>
<td>4214</td>
</tr>
</tbody>
</table>

Table 5.20. Comparison between IGATO and PaGMO's success rates on the GTOP Messenger Full problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>NFE (mil)</th>
<th>IGATO Success Rate</th>
<th>Equiv. #Trials</th>
<th>PaGMO Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>≤ 11</td>
<td>≤ 10</td>
<td>≤ 9</td>
</tr>
<tr>
<td>1</td>
<td>118</td>
<td>60%</td>
<td>40%</td>
<td>40%</td>
</tr>
<tr>
<td>2</td>
<td>383</td>
<td>80%</td>
<td>80%</td>
<td>40%</td>
</tr>
<tr>
<td>3</td>
<td>922</td>
<td>100%</td>
<td>100%</td>
<td>80%</td>
</tr>
<tr>
<td>4</td>
<td>2011</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>4214</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 5.7 shows the trajectory of best solution found by IGATO and figure 5.8 displays the Porkchop plot of the problem.
Figure 5.7. IGATO trajectory plot for the best found trajectory ($f=7.0584$ km/s) for the Messenger Full GTOP problem.

Figure 5.8. IGATO Porkchop plot of the Messenger Full GTOP problem.
V.III. Additional Test Cases

This section describes an additional test case which demonstrates IGATO's ability to perform round-trip surface stay missions. IGATO's performance is not compared to PaGMO or any other solver for this case because it is only for demonstrative purposes.

V.III.I. Case 4: Mars Round-Trip

For this problem, a round trip trajectory to the planet Mars is considered. In order to reduce the total ΔV, a Earth flyby is used before reaching Mars. Therefore the planetary sequence is Earth-Earth-Mars-Earth (EEME). Upon arriving at Mars, the spacecraft is desired to insert into a parking orbit with eccentricity $e=0.9$ and semimajor axis $a=37000km$. The spacecraft is allowed to stay in the parking orbit for a variable amount of time between 50 and 150 days. The problem has 16 dimensions and the state vector bounds are given below.
Table 5.21. The state vector bounds for the Mars Round-Trip problem.

<table>
<thead>
<tr>
<th>State</th>
<th>Variable</th>
<th>LB</th>
<th>UB</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1)</td>
<td>$t_0$</td>
<td>3650</td>
<td>7300</td>
<td>MJD2000</td>
</tr>
<tr>
<td>x(2)</td>
<td>$V_\infty$</td>
<td>1</td>
<td>5</td>
<td>km/sec</td>
</tr>
<tr>
<td>x(3)</td>
<td>$u$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(4)</td>
<td>$v_\theta$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(5)</td>
<td>$T_1$</td>
<td>50</td>
<td>1000</td>
<td>days</td>
</tr>
<tr>
<td>x(6)</td>
<td>$T_2$</td>
<td>50</td>
<td>1000</td>
<td>days</td>
</tr>
<tr>
<td>x(7)</td>
<td>$T_3$</td>
<td>50</td>
<td>1000</td>
<td>days</td>
</tr>
<tr>
<td>x(8)</td>
<td>$\eta_1$</td>
<td>0.01</td>
<td>0.99</td>
<td>days</td>
</tr>
<tr>
<td>x(9)</td>
<td>$\eta_2$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(10)</td>
<td>$\eta_3$</td>
<td>0.01</td>
<td>0.99</td>
<td>n/a</td>
</tr>
<tr>
<td>x(11)</td>
<td>$R_{p1}$</td>
<td>1.15</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>x(12)</td>
<td>$b_{incl1}$</td>
<td>$-\pi$</td>
<td>$\pi$</td>
<td>rad</td>
</tr>
<tr>
<td>x(13)</td>
<td>$T_{stay}$</td>
<td>50</td>
<td>150</td>
<td>days</td>
</tr>
<tr>
<td>x(14)</td>
<td>$V_{\infty,stay}$</td>
<td>0</td>
<td>3</td>
<td>km/s</td>
</tr>
<tr>
<td>x(15)</td>
<td>$u_{stay}$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>x(16)</td>
<td>$v_{stay}$</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
</tr>
</tbody>
</table>

IGATO was configured to run only 3 levels (4 subdomains on the final level) and the max NFE for each evolution was reduced back to 10,000 as with the Messenger and Cassini2 cases. The IGATO settings are listed below.
Table 5.22. The IGATO optimization settings for the Mars Round-Trip problem.

<table>
<thead>
<tr>
<th>IGATO Optimizer Settings: Messenger/Cassini2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels completed:</td>
</tr>
<tr>
<td>(4 subdomains in the final level)</td>
</tr>
<tr>
<td>Number of evolutions per subdomain:</td>
</tr>
<tr>
<td>Max NFE per evolution:</td>
</tr>
<tr>
<td>Dynamic restart threshold:</td>
</tr>
<tr>
<td>Starting population size:</td>
</tr>
<tr>
<td>Ending population size:</td>
</tr>
<tr>
<td>Number of intervals per decision variable:</td>
</tr>
<tr>
<td>Total % of each decision variable pruned:</td>
</tr>
<tr>
<td>Similarity Tests:</td>
</tr>
</tbody>
</table>

The results of IGATO are listed below along with trajectory plot of the best found solution (Figure 5.9) and the Porkchop plot of the problem (Figure 5.10). As usual the trajectory details can located Appendix A.

Table 5.23. IGATO results for the Mars Round-Trip problem.

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of Subdomains Completed</th>
<th>NFE required (millions)</th>
<th>Best Solution (km/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This Level</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>91.3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>308.6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>704.2</td>
</tr>
</tbody>
</table>
Figure 5.9. IGATO trajectory plot for the best found solution ($f=6.3065$ km/s) for the Mars Round-Trip problem.

Figure 5.10. IGATO Porkchop plot for the Mars Round-Trip problem.
VI. Software

This chapter introduces the software implementation of IGATO. The chapter begins with a brief overview of some of the design choices made and then provides an in-depth user guide for using the software.

VI.I. Design Choices

When it comes to software implementation, there are often many choices to make. This section will cover some of the major design choices made during implementation:

- What is the target platform(s)? Windows, Mac OS X, Linux, etc.
- How is it licensed? Commercial, GPL, LGPL, etc.
- What language to write it in? Matlab, Fortran, C, C++, C#, Python, etc.
- How to create the graphical user interface (GUI)? MATLAB, WPF, Windows Forms, Qt, etc.

Considering that the design objectives of this work was to create an open-source cross-platform application, the first design choice was easy. The target platform(s) would be all three major operating systems: Windows, Mac OS X, and Linux. For licensing, since PaGMO uses the GPL license, this application had no choice other than use the GPL license as well. Any software that uses a library/application protected under the GPL must itself use the GPL license. The GPL license enforces that the source code must be
available to anybody who receives the application. For the primary programming language, this software chose to C++ and is written in an object-oriented style. This choice was made over other languages prevalent in scientific computing such as C or Fortran90 which generally offer greater performance, because C++ offers better object oriented capabilities. This was also largely a matter of personal opinion, the author already had some experience in C++ and preferred the programming language and style compared to C and Fortran90. Despite being slightly slower than C or Fortran, C++ is much faster than interpreted languages like Python and Matlab. Since global optimization is computationally demanding, the performance gain of C++ was chosen over the interpreted alternatives despite their ease of use and much faster development time. The only language which really was a contender was C# which is considered to be an evolved form of C++ and does away with many of its nuances. Developing in C# tends to be much faster than C++ and the performance between the two is comparable, with C++ typically being slightly faster. However, C# is a .NET language created and closely tied to Microsoft. Although there are ways to port C# to other platforms, the author chose to stick with C++ due to previous experience and simplicity. C++ has been one of the central programming languages in many industries for a number of years which has lead to numerous libraries, references, and support online. C++ is also the primary language of the Qt class library which was chosen to implement the graphical user interface. Qt is a mature and powerful library which has a strong object oriented and platform independent mentality. Since Nokia's acquisition of Trolltech in 2008, Qt has seen a surge of rapid development and growth and is now arguably one of the best GUI libraries currently available, with a strong future still in front of it. There are also many
useful extensions for Qt such as QwtPlot which was used for all the plotting in this software.

This section walks through IGATO's graphical user interface and introduces many of the application's robust capabilities.

**VI.II. IGATO GUI User Guide**

This section provides a full user guide to using IGATO.

**VI.II.I. Trajectory Inputs**

Upon starting up IGATO, the user is greeted with the main window with the **Trajectory Inputs** tab selected. This tab allows the user to specify a wide range of interplanetary trajectory scenarios. To begin, specify where the spacecraft will be starting its journey by clicking the button labeled *Add Departure*. 
Departure Object

Clicking the *Add Departure* button adds a departure object to the *Interplanetary Itinerary* viewer on the left side of the main window. Selecting this item loads a pane on the right side of the main window which allows the user to specify different options regarding the departure object.

- **Departure Object** - Departure object name. Any planet can be selected and user-defined objects can be created by selection *custom*.
• **Ephemeris** - Type of ephemeris data used for propagation of orbital bodies. There are two options: *JPL* and *File*.
  
  o **JPL** - Ephemeris is generated by using data from NASA's JPL database. Selecting JPL allows two options: approximate *Analytic* propagations using algorithms and constants published by JPL or propagation based on observational data stored in JPL's DE405 file. Analytic propagations are generally faster whereas DE405 is typically more accurate.
  
  o **File** - If observational propagation is desired for a time frame that is not supported by IGATO's DE405 file, a user-supplied file can be used instead. The format of this file must be the same as DE405. JPL often changes the formatting style over time.

• **Parking Orbit** - This option allows the spacecraft to begin its trajectory from an initial parking orbit instead of from the surface of the departure orbit. Enabling this option requires the user to specify the 2D orbital elements of the parking orbit.
  
  o **Semimajor Axis** - Semimajor axis of parking orbit (km).
  
  o **Eccentricity** - Eccentricity of parking orbit.

• **Departure Velocity Limit** - The minimum and maximum velocity allowed while departing from the Departure Object. By default the format of this parameter is DeltaV with units of kilometers per second (km/s). The second option is to use the C3 energy with units of kilometers squared per second squared (km^2/s^2).
Arrival Object

The *Arrival Object* can be added to the *Interplanetary Itinerary* viewer, after the *Departure object* has been added. Selecting this object changes the pane on the right side of the main window to allow customization of the arrival object's different options.

- **Arrival Object** - Arrival object name. Any planet can be selected and user-defined objects can be created by selection *custom*.

- **Ephemeris** - Type of ephemeris data used for propagation of orbital bodies. There are two options: *JPL* and *File*. 
- **JPL** - Ephemeris is generated by using data from NASA's JPL database. Selecting JPL allows two options: approximate *Analytic* propagations using algorithms and constants published by JPL or propagation based on observational data stored in JPL's DE405 file. Analytic propagations are generally faster whereas DE405 is typically more accurate.

- **File** - If observational propagation is desired for a time frame that is not supported by IGATO's DE405 file, a user-supplied file can be used instead. The format of this file must be the same as DE405. JPL often changes the formatting style over time.

- **Time of Flight** - The minimum and maximum time (days) that it takes to reach the next orbital object.

- **Insertion Orbit** - This option allows the spacecraft to insert into a final parking orbit around the Arrival Object. Enabling this option requires the user to specify the 2D orbital elements of the insertion orbit.
  - **Semimajor Axis** - Semimajor axis of insertion orbit (km).
  - **Eccentricity** - Eccentricity of insertion orbit.

- **Deep Space Maneuver(s)** - How many deep space maneuvers are allowed between the previous orbital object and the Arrival Object.
  - **Alpha** - The fraction along the arc from the previous object to the Arrival Object at which the DSM occurs. If more than one DSM is allowed, subsequent alpha's after the first are applied to the time of flight remaining along the arc.
  - **Delta-v** - If more than one DSM is allowed, this parameter determines the maximum delta-v that is allowed during all DSM's excluding the last one.
Flyby Object

One or multiple *Flyby Objects* can be added to the *Interplanetary Itinerary* viewer, after the *Arrival Object* has been added. Selecting this object changes the pane on the right side of the main window to allow customization of the flyby object's different options.

- **Flyby Object** - Flyby object name. Any planet can be selected and user-defined objects can be created by selection *custom*.

- **Ephemeris** - Type of ephemeris data used for propagation of orbital bodies. There are two options: *JPL* and *File*.

Figure 6.3. The Trajectory Inputs tab with an arrival object selected.
- **JPL** - Ephemeris is generated by using data from NASA's JPL database. Selecting JPL allows two options: approximate *Analytic* propagations using algorithms and constants published by JPL or propagation based on observational data stored in JPL's DE405 file. Analytic propagations are generally faster whereas DE405 is typically more accurate.

- **File** - If observational propagation is desired for a time frame that is not supported by IGATO's DE405 file, a user-supplied file can be used instead. The format of this file must be the same as DE405. JPL often changes the formatting style over time.

- **Time of Flight** - The minimum and maximum time (days) that it takes to reach the next orbital object.

- **Flyby Distance** - The minimum and maximum distance allowed during the flyby. Two different formats are allowed: The default format is *Altitude* which is the distance from the flyby object's mean geometric radius. The second option is *Planet Radius* which is the distance from the flyby object's center. Both formats expect values with units of kilometers (km).

- **Surface Stay Orbit** - This option allows the spacecraft to insert into a surface stay orbit for a specified amount of time upon reaching the flyby object instead of passing by it. Enabling this option requires the user to specify the 2D orbital elements of the surface stay orbit.
  - **Semimajor Axis** - Semimajor axis of surface stay orbit (km).
  - **Eccentricity** - Eccentricity of surface stay orbit.
  - **Duration** - The minimum and maximum length of time (days) to stay at the flyby object.
• **Deep Space Maneuver(s)** - How many deep space maneuvers are allowed between the previous orbital object and the current Flyby Object.

  o **Alpha** - The fraction along the arc from the previous orbital object to the current Flyby Object at which the DSM occurs. If more than one DSM is allowed, subsequent alpha's after the first are applied to the time of flight remaining along the arc.

  o **Delta-v** - If more than one DSM is allowed, this parameter determines the maximum delta-v that is allowed during all DSM's excluding the last one.

---

**Figure 6.4. The Trajectory inputs tab with a flyby object selected.**
VI.II.II.  Optimizer Settings

The optimizer settings tab allows full configuration of the PaGMO archipelago (right panel) and the additional optimization strategies (left panel).

- **PaGMO Archipelago Settings** - this pane configures the archipelago.
  
  - **New/Save/Save As** - PaGMO archipelago settings can be saved and reused by saving them as a .PA file.
  
  - **Number of Islands** - How many islands are in the archipelago.
  
  - **Topology** - The island topology. PaGMO comes equipped with numerous options.
  
  - **Number of Migrations (deprecated)** - The number of migrations that are allowed to occur during an evolution.
  
  - **Select Island** - The index of the island the following options will apply to.
  
  - **Population Size (deprecated)** - The population size of the selected island.
  
  - **Selection Policy** - The selection policy of the selected island. The selection policy determines which members of the population are migrated to other islands.
  
  - **Replacement Policy** - The replacement policy of the selected island. The replacement policy determines how an island chooses to accept incoming migrations.
  
  - **Select Algorithm** - The algorithm which will inhabit the island.
  
  - **Configure *Algorithm*** - The parameters of the selected algorithm. These parameters vary depending on which algorithm is selected.
• **General** - The high-level settings for the optimizer.
  
  o **New/Save/Save As** - Optimization settings can be saved and reused by saving them as a .OPT file. A .OPT file consists of the general optimization settings in addition to the PaGMO settings.

  o **Stopping Criteria** - The overall stopping criteria of the optimization. The stopping criteria can be specified as either a time (minutes) or a number of function evaluations.

  o **Max Subdomains** - The maximum number of active subdomains allowed. The optimizer always starts with one subdomain (the domain) and bisects a subdomain after every evolution loop until the number of subdomains reaches the maximum allowed.

  o **Similarity Tests** - This option toggles on or off similarity tests. By checking this option, the optimizer will perform the similarity tests on the best solution of the current subdomain and the best solution of the domain after each evolution loop.

  o **Restarts** - These options are for configuring static or dynamic restarts for each evolution in the evolution loop.
    
    ▪ **Static** - This option toggles on or off static restarts. A static restart ends an evolution immediately once the stopping criteria is reached. The stopping criteria can either be time (minutes) or number of function evaluations. In addition, a minimum and maximum stopping criteria can be set in which the stopping criteria
for each evolution is chosen at random between the minimum and maximum values.

- **Dynamic** - This option toggles on or off dynamic restarts. A dynamic restart ends an evolution once the average range of each coordinate drops below the difference threshold.

  - **Evolution Loop** - These options are for configuring the evolution loop.
    
    - **Number of Evolutions** - The number of evolutions performed for each evolution loop.
    
    - **Initial/Final Pop** - For the first evolution of the evolution loop, the population size of each island is set to the initial pop. The population size increases/decreases after each evolution in a linear manner such that the last evolution has a population size equal to the final pop.

  - **Pruning** - This check box toggles on or off the pruning algorithm. Checking this box enables the pruning algorithm which prunes intervals after each evolution in the evolution loop.
    
    - **Num. Intervals** - The number of intervals each coordinate is divided into in order to archive solutions. It's recommended to use about 100 intervals for best performance.
    
    - **Intervals Pruned** - The number of intervals pruned throughout the entire evolution loop. This number can be equal to or greater than Num. Intervals because pruning an interval results in the bisection of another (i.e. there is always a fixed number of intervals).
- **Prune from previous evolutions** - This option toggles on or off the ability for the pruning algorithm to learn from previous evolution loops. Enabling this option guarantees accumulation of knowledge throughout the optimization. Disabling this option clears the pruning algorithm's knowledge of the search space after an evolution loop is finished.

![Image of the Optimizer Inputs tab]

Figure 6.5. The Optimizer Inputs tab.

VI.II.III. Output

Once the interplanetary trajectory and the optimizer are both setup, the optimization can be run by clicking the *Start* button located at the top of the main window to the right of
the IGATO logo. Starting the optimization, automatically selects the **Output** tab of the main window. From this tab, the user can use a variety of tools to overlook the progress of the optimization routine and analyze the solutions that are found during run time. The **Output** tab consists of several buttons, two different trajectory lists for selecting promising solutions, and the output pane with four sub-tabs: **Simulation Outputs**, **Solution Summary**, **Trajectory Details**, and **Trajectory Plot**.

**Buttons**

Several buttons are located in the upper left corner below the IGATO logo. These buttons are listed below:

- **Realtime Plots** - This button launches a separate window which allows monitoring the ongoing evolutions on each island in the archipelago. See Section VI.II.V.

- **Porkchop Plot** - This button launches a separate window which can be used to created a Porkchop plot that is updated in realtime as the optimization progresses. See Section VI.II.VI.

- **Open Outputs** - This button opens the directory **IGATO/Output** which holds various files summarizing the trajectory and optimizer settings, and the optimizer results. See Section VI.II.VII.

**Best Domain & Subdomain Trajectories**

The two lists on the far left side display the best solutions found to date during the current optimization. The list labeled **Best Domain Trajectory** displays the best solution found
within the entire search space (domain). The list labeled *Best Subdomain Trajectories* displays the best solutions found within each subdomain given that *Subdomain Decomposition* was enabled in the optimization settings. Clicking on these list items gives detailed trajectory information in the *Trajectory Details* tab and plots/simulates the trajectory in the *Trajectory Plot* tab as discussed below.

**Output Pane**

The output pane consists of four sub-tabs: *Simulation Outputs*, *Solution Summary*, *Trajectory Details*, and *Trajectory Plot*. These tabs provide an overview of the optimization underway and also provide detailed information for specific trajectories that are selected.

- **Simulation Outputs** - This tab outputs the solution after each evolution in a streaming text format. After each evolution the, evolution number, best objective function value, state vector corresponding to the best objective function value, number of function evolutions, and run time are displayed for that evolution.

- **Solution Summary** - This tab displays most of the same information as the Simulation Outputs tab, but in tabular form.

- **Trajectory Details** - This tab gives an in-depth overview of a trajectory selected from the *Best Domain Trajectory* or *Best Subdomain Trajectory* lists. First, a general overview of the mission is outlined. This information includes the sequence of orbital objects visited along the trajectory, the departure and arrival dates, and the total required velocity and flight time of the mission. Below the mission overview is a more detailed breakdown of each major event that occurs
throughout the trajectory. An event can either be departure, arrival, a flyby, or a DSM. For each event relevant information such as the date, and required change in velocity is given.

- **Trajectory Plot** - This tab displays a 2D projection of the full spacecraft trajectory onto the orbital plane along with any orbital bodies that are part of the trajectory. In addition, an ellipse for each planet/orbital body that the spacecraft visits throughout the trajectory is displayed as well. The spacecraft's trajectory is colored in green whereas each planet/orbital body has its own distinct color. Initially, the plot displays the trajectory at the end of the mission. The position of each planet/orbital body at the end of the mission is represented by a solid circle with the name of the orbital body above it. The departure and arrival events are also marked with a gray diamond with the words "Departure" and "Arrival" under them respectively.

  - **Trajectory Simulation** - The trajectory plot can also be used to simulate the trajectory using the six controls located at the top-center of the plot. From left to right, the "<|" button resets the trajectory to the beginning of the mission. The "<<" and ">>" buttons speed up and slow down the simulation respectively. The "<" and ">" buttons rewind and fast forward the trajectory to the previous and next events respectively. Lastly, the "|>" button starts the simulation. Once the "|>" is pressed, the icon changes to "||" which pauses the simulation. During a simulation, selecting other solutions from the Best Domain Trajectory or Best Subdomain Trajectories lists will have no effect.
o **Zoom** - The trajectory plot can also be zoomed in by creating a zoom-box.

To do so, hold down the left mouse button and move the mouse. Release the left mouse button to zoom in on the selected area. To zoom out one time, click the right mouse button. To reset the zoom to the initial state, hold down the SHIFT key and click the right mouse button.

![Image of IGATO interface with Simulation Outputs sub-tab highlighted]

**Figure 6.6.** The Simulation Outputs sub-tab of the Output tab.
Figure 6.7. The Solution Summary sub-tab of the Output tab.

Figure 6.8. The Trajectory Details sub-tab of the Output tab.
VI.II.IV. Real time Archipelago Analysis

One of the extra useful utilities that IGATO is equipped with is the Real time Archipelago Analysis window (RTAA). This window grants the ability to monitor the state of ongoing evolutions within the archipelago in real time and is accessed by clicking the Real time Plots button on the Output tab of the main window. Clicking this button launches a separate window which contains three tabs: Population Breakdown, Cost History, and Solution Summary. The first two tabs characterize the state of the current
evolution underway, while the third tab summarizes the solutions found after each evolution. The purpose of these items is detailed below:

**Top Level Controls**

At the top of the window, there are located four controls: "Start" button, "Sample Rate" spin box, drop down list labeled "Select Trial", and "Insert Best" button.

- **Start Button** - This button enables/disables collecting data for real time plotting. Clicking "Start" begins requesting data from the Archipelago at the specified sample rate (see below). Data collection allows real time plotting, but can also slow down the optimization program. Clicking "Stop" disables data collection entirely which places no extra computational effort on the optimizer.

- **Sample Rate** - This is the frequency at which data is requested from the Archipelago for real time plotting. A suggested sample rate is 1-5 Hz depending on the system's hardware. Increasing the sample rate too much can lead to excessive slow down. After entering a desired sample rate, press ENTER or click on a different control to set the sample rate.

- **Select Trial** - This drop down list selects which evolution is being analyzed. By default, "Current" is selected which always uses the current evolution. Past evolutions can be chosen instead by selecting them from the drop down list.

- **Insert Best Button** - This button opens a small dialog which allows the user to input a solution to be plotted along with the current population. See the **Inserting a Known Solution** section below.

•
Plot Tabs

The bulk of information displayed by the RTAA is located in the three central tabs. These tabs are described below:

- **Population Breakdown Tab** - This tab displays information about the state vector of each member of the population of the specified trial (set in the "Select Trial" drop down list). On the right are two stacked plots labeled "Single Coordinate Plot" and "Parallel Coordinate Plot" and on the left are two group boxes with various options for each plot respectively.

  - **Single Coordinate Plot** - This plot displays the cost associated with a particular coordinate of the state vector for all members of the population. The cost is plotted on the y-axis and the selected coordinate is plotted on the x-axis. Several options located in the *Single Coordinate Plot Options* group box to the left of the plot allow further customization:

    - **Select Coordinate** - This dropdown list allows the user to select different coordinate to be plotted.

    - **Fixed Coordinate Axis (x-axis)** - Checking this option fixes the x-axis to the full range of the selected coordinate. Unchecking this option reduces the x-axis to the minimum range that encapsulates all population members.

    - **Max Cost (y-axis)** - Checking this option allows the user to specify a max value (cost) for the y-axis in the text box to the right. All population members with a cost above the max value specified
are ignored. Checking the checkbox labeled "Auto" automatically sets the maximum cost to 1.5x the minimum cost.

- **Parallel Coordinate Plot** - This is a type of plot that is commonly used to visualize multidimensional datasets. The plot is characterized by a number of vertical black lines with numbers below and above each one, along with a series of blue lines connecting each vertical line to the one next to it. Here each the vertical line represents a coordinate of the state vector. For instance, the first vertical line is always the departure date (usually in MJD2000). The numbers below and above the vertical coordinate line are the minimum and maximum values respectively that were specified in the **Trajectory Inputs** tab. A solution then has a vertex on each vertical coordinate, and can be visualized by connecting each consecutive vertex with a blue line. Several options located in the **Parallel Coordinate Plot Options** group box to the left of the plot allow further customization:
  - **Show Unit Labels** - This option shows or hides the minimum and maximum coordinate values below and above the vertical coordinate lines respectively.
  - **Transparency** - The transparency can be adjusted so that it is easier to identify trends when multiple solutions are plotted simultaneously.
  - **Shown & Not Shown** - Individual coordinates can be added or removed from the plot by moving them between the "Shown" and
"Not Shown" lists respectively. Coordinates can also be reorganized by moving them up or down in the "Shown" list.

![Image of the Population Breakdown tab](image.png)

**Figure 6.10. The Population Breakdown tab.**

- **Objective Function History Tab**

  This tab consists of a single plot which tracks the best solutions found on each island in the archipelago. At each sample point, the objective function associated with the best solution currently living on each island is plotted in a different color. A legend is located on the right of the plot to identify which islands corresponds to which lines. Sometimes an island's best objective function drops dramatically over a very short time. This is either the result of the island uncovering a new
promising region of the search space or the sign that the island recently received a much better solution from another island through the migration mechanism. By default the plot is limited to displaying a maximum of 200 sample points. Once this number is reached, the plot begins to scroll: Newer points are added while older points are removed. After each evolution, the history is reset. The plot can also be zoomed by using the same commands described in the Trajectory Plot section. Several options located in the Options group box below the plot allow further customization:

- **Select Island** - Selects which island the subsequent options are applied to.
- **Line Width** - Sets the line width of the selected island.
- **Fixed x-axis** - Fixes the x-axis to the minimum and maximum values specified in the fields below.
- **Max y-axis** - Sets the maximum objective function value for the y-axis. Islands with a best solution exceeding this value are not plotted. Specifying "Auto" sets the max objective function value to 1.5x the minimum objective function value.
- **Shown & Not Shown** - Individual islands can be added or removed from the plot by moving them between the "Shown" and "Not Shown" lists respectively.
Solution Summary Tab

This tab looks identical to the **Population Breakdown Tab**. Unlike the first tab which characterizes the selected population, this tab characterizes the best solutions found after each evolution. It is therefore not affected by the sample rate or currently selected trial.
Inserting a Known Solution

If the problem being optimized has a known global optima or best known solution, this information can be overlayed on top of the archipelago plots for comparison. To do so, click the "Insert Best" button in the upper right corner of the window. This button opens a small dialog which asks for the objective function value and state vector of the solution to be inserted. It is important to make sure the state vector variables are in the same order of the state vector being used in the optimization.
Figure 6.13. The Best Solution Dialog pop-up window.

By clicking OK, after the two fields are filled in, the inserted solution is plotted as a red diamond in all the single dimension plots and as a bold red line in all the parallel coordinate plots.

Figure 6.14. The Population Breakdown tab with a best known solution inserted (red).
VI.II.V. Real Time Porkchop Plot

Porkchop Plots are typically used to identify optimal transfer arcs between two points given a range of departure times and a range of flight times (or arrival times). A Porkchop Plot expresses 3-dimensional data on a 2-dimensional plot using either a contour or spectrogram type plot. Typically, the departure time is on the x-axis, the flight time is on the y-axis and the z-axis (delta-v, objective function, etc.) is visualized in 2D using the contour or spectrogram. Porkchop Plots are easily generated for simple transfer arcs between two objects because there are only two independent variables which can be discretized and enumeratively combined. Using a Porkchop Plot to visualize a MGA-DSM trajectory is more difficult because such discretization and enumeration over all state variables is unfeasible, hence the need for heuristics. Therefore a Porkchop Plot for a MGA-DSM trajectory using heuristics is only an approximation of the real thing. IGATO can generate an approximate Porkchop Plot by first clicking the "Porkchop Plot" button on the Output tab of the main window. This launches a separate window with a few options near the top and an empty plot. The options at the top of the window are detailed below:

- **Start Button** - This button is located in the upper left corner of the window. By clicking "Start", the optimizer will begin archiving the best objective function values belonging to each discretized departure date and flight time and plotting them in the Porkchop Plot. The fidelity* of the discretization can be specified in the Optimizer Settings tab and cannot be altered while an optimization is underway.
- **Show Contour** - Show or hide the contour lines.
- **Show Spectrogram** - Show or hide the spectrogram.
- **Max Z** - Checking this option sets the maximum z-axis of the plot to the value in the spin box to the right.

![Porkchop Plot Dialog](image)

**Figure 6.15.** IGATO generated Porkchop plot of the Cassini 2 problem from the GTOP database.

### VI.II.VI. Output Files

IGATO outputs several useful files during and after execution. These files summarize the trajectory inputs, optimizer settings, and optimization results. These files are stored in the IGATO/Output directory and are described below:
Output

During optimization, this file is continuously updated with information describing the best solutions found after each evolution. For each evolution, the objective function value (cost), number of function evaluations (NFE), time in milliseconds, and state vector is given.

Figure 6.16. The Output file generated by IGATO.

*Options are not fully implemented at the time of writing.
VII. Conclusion

VII.I. Summary of Work

This work attempted to explore the world of global optimization applied to MGA-DSM trajectory optimization. In the end, a robust high-level, and extendable optimization routine was created and tested on several challenging MGA-DSM problems. The optimization routine was built around the open-source optimizer PaGMO which is an implementation of the Island Model Paradigm developed by ESA's Advanced Concepts Team (ACT). This work augmented PaGMO with a variety of high level strategies such as dynamic restarts to prevent stagnation, a pruning algorithm which grows increasingly smarter the longer it is allowed to run, subdomain decomposition which promotes finding good solutions for multiple launch windows, and a new testing procedure which aims to take advantage of the similarity of good solutions. The optimizer performed very well, consistently finding the best known solutions to difficult MGA-DSM problems from the GTOP database as fast or faster than PaGMO could in the same number of function evaluations. The exception was the Messenger Full problem, which neither IGATO or PaGMO fared very well against. The best solution discovered by IGATO was well above the best known solution, however it was still able to find a much better solution than PaGMO.
VII.II. Future Work

The work has the potential to grow in many different directions. The optimization routine needs more research and testing to help solidify whether all of its elements are truly beneficial. In particular, the Similarity Testing needs more testing. I suspect that the first of the three tests is the most useful whereas the latter two are slightly redundant and could be altered or removed entirely to increase efficiency. The trajectory model and GUI have plenty of room to grow as well. The following lists some of the major areas IGATO could be improved upon. Luckily, the first three items in the list are already supported in PaGMO.

- Mixed integer continuous optimization support
  - Support for integer and continuous state variables allows the trajectory model to more easily handle multi-rev Lambert solutions and variable planet sequences.

- Multiobjective optimization support
  - This work concentrated on minimizing $\Delta\ell$, but in reality flight time is equally important, especially for manned-missions. Multiobjective optimization with Pareto-fronts could be used to optimize both criterion simultaneously.

- Support for local optimization libraries (e.g. SNOPT and NLopt)
  - This work primarily utilized global optimizer metaheuristics, but local optimizers could be included as well which help refine desirable solutions.

- User-defined bodies (asteroids, etc.)
Many interplanetary trajectories involve flybys of asteroids, and as such, the GUI should support the ability for the user to define a custom celestial body.

- Ability to shift to and from planetocentric coordinate system
  - This work concentrated on the heliocentric coordinate system (interplanetary), but the ability to shift into a planetocentric coordinate system could be useful in case a trajectory desired to visit a moon around a planet, etc.
Appendix. Trajectory Details

This appendix lists the detailed trajectory information for the best found solutions found by IGATO for each of the test cases from Chapter V.

**Messenger**

Mission overview and breakdown:

---

**Mission Overview**

<table>
<thead>
<tr>
<th>Sequence:</th>
<th>EEVVMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departure Date:</td>
<td>Sun March 16th, 2003 18:30:20</td>
</tr>
<tr>
<td>Arrival Date:</td>
<td>Wed February 08th, 2006 01:55:36</td>
</tr>
<tr>
<td>Total C3:</td>
<td>74.4743 km2/s2</td>
</tr>
<tr>
<td>Total DeltaV:</td>
<td>8.62985 km/s</td>
</tr>
<tr>
<td>Total TOF:</td>
<td>1059.31 days</td>
</tr>
</tbody>
</table>
---

**Mission Breakdown**

| Departure Object: | Earth |
| Departure Date: | Sun March 16th, 2003 18:30:20 |
| Departure Julian Date: | 2452715.7711 |
| Departure DeltaV: | 1.42701 km/s |
| Departure C3: | 2.03635 km2/s2 |
| DSM 1 Date: | Thu June 19th, 2003 10:54:19 |
| DSM 1 Julian Date: | 2453115.7711 |
| DSM 1 DeltaV: | 0.908207 km/s |
| DSM 1 C3: | 0.82484 km2/s2 |
| DSM 1 TOF: | 94.6833 days |
| Flyby 1 Object: | Earth |
| Flyby 1 Date: | Mon April 19th, 2004 18:30:20 |
| Flyby 1 Julian Date: | 2453115.7711 |
| Flyby 1 DeltaV: | 0 km/s |
| Flyby 1 C3: | 0 km2/s2 |
| Flyby 1 TOF: | 305.317 days |
| DSM 2 Date: | Wed June 09th, 2004 02:18:04 |
| DSM 2 Julian Date: | 2453295.186 |
| DSM 2 DeltaV: | 0 km/s |
| DSM 2 C3: | 0 km2/s2 |
| DSM 2 TOF: | 50.3248 days |
| Flyby 2 Object: | Venus |
| Flyby 2 Date: | Sat October 16th, 2004 04:27:49 |
---
Flyby 2 Julian Date: 2453295.186
Flyby 2 ΔV: 0 km/s
Flyby 2 C3: 0 km²/s²
Flyby 2 TOF: 129.09 days

DSM 3 Date: Wed June 22nd, 2005 04:31:12
DSM 3 Julian Date: 2453594.4672
DSM 3 ΔV: 0.248367 km/s
DSM 3 C3: 0.0616863 km²/s²
DSM 3 TOF: 249.002 days

Flyby 3 Object: Venus
Flyby 3 Date: Thu August 11th, 2005 11:12:46
Flyby 3 Julian Date: 2453594.4672
Flyby 3 ΔV: 0 km/s
Flyby 3 C3: 0 km²/s²
Flyby 3 TOF: 50.2789 days

DSM 4 Date: Thu October 06th, 2005 21:50:09
DSM 4 Julian Date: 2453775.0803
DSM 4 ΔV: 1.43822 km/s
DSM 4 C3: 2.06847 km²/s²
DSM 4 TOF: 56.4426 days

Arrival Object: Mercury
Arrival Date: Wed February 08th, 2006 01:55:36
Arrival Julian Date: 2453775.0803
Arrival ΔV: 4.60805 km/s
Arrival C3: 21.2341 km²/s²
Arrival TOF: 124.17 days

State Vector:
1171.29, 1.41663, 0.499516, 0.236275, 400, 1.76637, 1.35113, 0.0412035, 178.735, 3.03443, 1.09395, 0.833352, 299.271, 1.1, 1.3443, 0.312578, 180.723

**Cassini 2**

Mission overview and breakdown:

-----------------------------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Mission Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence: EVVEJS</td>
</tr>
<tr>
<td>Departure Date: Tue November 11th, 1997 20:39:50</td>
</tr>
<tr>
<td>Arrival Date: Mon April 09th, 2007 09:40:35</td>
</tr>
<tr>
<td>Total C3: 70.2754 km²/s²</td>
</tr>
<tr>
<td>Total ΔV: 8.38304 km/s</td>
</tr>
<tr>
<td>Total TOF: 3435.54 days</td>
</tr>
</tbody>
</table>

-----------------------------------------------------------------------------------------------
### Mission Breakdown

<table>
<thead>
<tr>
<th>Departure Object</th>
<th>Departure Date</th>
<th>Departure Julian Date</th>
<th>Departure DeltaV</th>
<th>Departure C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>Tue November 11th, 1997 20:39:50</td>
<td>2450764.861</td>
<td>3.26879 km/s</td>
<td>10.685 km2/s2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSM 1 Date</th>
<th>DSM 1 Julian Date</th>
<th>DSM 1 DeltaV</th>
<th>DSM 1 C3</th>
<th>DSM 1 TOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun March 22nd, 1998 18:20:14</td>
<td>2450933.1933</td>
<td>0.469182 km/s</td>
<td>0.220132 km2/s2</td>
<td>130.903 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flyby 1 Object</th>
<th>Flyby 1 Date</th>
<th>Flyby 1 Julian Date</th>
<th>Flyby 1 DeltaV</th>
<th>Flyby 1 C3</th>
<th>Flyby 1 TOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Venus</td>
<td>Wed April 29th, 1998 04:38:22</td>
<td>2450933.1933</td>
<td>0 km/s</td>
<td>0 km2/s2</td>
<td>37.4293 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSM 2 Date</th>
<th>DSM 2 Julian Date</th>
<th>DSM 2 DeltaV</th>
<th>DSM 2 C3</th>
<th>DSM 2 TOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sat December 12th, 1998 11:16:10</td>
<td>2451357.3363</td>
<td>0.39837 km/s</td>
<td>0.158698 km2/s2</td>
<td>227.276 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flyby 2 Object</th>
<th>Flyby 2 Date</th>
<th>Flyby 2 Julian Date</th>
<th>Flyby 2 DeltaV</th>
<th>Flyby 2 C3</th>
<th>Flyby 2 TOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Venus</td>
<td>Sun June 27th, 1999 08:04:16</td>
<td>2451357.3363</td>
<td>0 km/s</td>
<td>0 km2/s2</td>
<td>196.867 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSM 3 Date</th>
<th>DSM 3 Julian Date</th>
<th>DSM 3 DeltaV</th>
<th>DSM 3 C3</th>
<th>DSM 3 TOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sat July 03rd, 1999 15:46:33</td>
<td>2451410.644</td>
<td>0 km/s</td>
<td>0 km2/s2</td>
<td>6.32103 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flyby 3 Object</th>
<th>Flyby 3 Date</th>
<th>Flyby 3 Julian Date</th>
<th>Flyby 3 DeltaV</th>
<th>Flyby 3 C3</th>
<th>Flyby 3 TOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>Thu August 19th, 1999 15:27:18</td>
<td>2451410.644</td>
<td>0 km/s</td>
<td>0 km2/s2</td>
<td>46.9866 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSM 4 Date</th>
<th>DSM 4 Julian Date</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon January 10th, 2000 17:30:32</td>
<td>2452000.4148</td>
<td></td>
</tr>
</tbody>
</table>
DSM 4 DeltaV: 0 km/s
DSM 4 C3: 0 km2/s2
DSM 4 TOF: 144.086 days

Flyby 4 Object: Jupiter
Flyby 4 Date: Sat March 31st, 2001 09:57:15
Flyby 4 Julian Date: 2452000.4148
Flyby 4 DeltaV: 0 km/s
Flyby 4 C3: 0 km2/s2
Flyby 4 TOF: 445.685 days

DSM 5 Date: Tue August 22nd, 2006 10:19:43
DSM 5 Julian Date: 2454200.4032
DSM 5 DeltaV: 0 km/s
DSM 5 C3: 0 km2/s2
DSM 5 TOF: 1970.02 days

Arrival Object: Saturn
Arrival Date: Mon April 09th, 2007 09:40:35
Arrival Julian Date: 2454200.4032
Arrival DeltaV: 4.2467 km/s
Arrival C3: 18.0345 km2/s2
Arrival TOF: 229.973 days

State Vector:
-780.201, 3.26939, 0.578676, 0.383501, 0.773612, 168.375, 1.34899,
-1.59232, 0.535724, 424.164, 1.05, -1.95953, 0.228769,
53.3069, 1.30677, -1.55477, 0.18925, 589.772, 69.8066, -1.51343, 0.895458, 2200

**Messenger Full**
Mission overview and breakdown:

---------------------------------------------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Mission Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence: EEVVMeMeMe</td>
</tr>
<tr>
<td>Departure Date: Sat March 25th, 2006 06:01:09</td>
</tr>
<tr>
<td>Arrival Date: Sat August 14th, 2010 05:54:28</td>
</tr>
<tr>
<td>Total C3: 49.8212 km2/s2</td>
</tr>
<tr>
<td>Total DeltaV: 7.05841 km/s</td>
</tr>
<tr>
<td>Total TOF: 1603 days</td>
</tr>
</tbody>
</table>

---------------------------------------------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Mission Breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departure Object: Earth</td>
</tr>
<tr>
<td>Departure Date: Sat March 25th, 2006 06:01:09</td>
</tr>
<tr>
<td>Departure Julian Date: 2453820.2508</td>
</tr>
<tr>
<td>Departure DeltaV: 2.5 km/s</td>
</tr>
<tr>
<td>Departure C3: 6.25 km2/s2</td>
</tr>
</tbody>
</table>
DSM 1 Date:               Sat November 04th, 2006 05:46:15
DSM 1 Julian Date:        2454313.729
DSM 1 DeltaV:             0.624729 km/s
DSM 1 C3:                 0.390286 km2/s2
DSM 1 TOF:                223.99 days

Flyby 1 Object:           Earth
Flyby 1 Date:             Tue July 31st, 2007 17:29:44
Flyby 1 Julian Date:      2454313.729
Flyby 1 DeltaV:           0 km/s
Flyby 1 C3:               0 km2/s2
Flyby 1 TOF:              269.489 days

DSM 2 Date:               Thu September 13th, 2007 10:46:14
DSM 2 Julian Date:        2454484.5467
DSM 2 DeltaV:             0 km/s
DSM 2 C3:                 0 km2/s2
DSM 2 TOF:                43.7198 days

Flyby 2 Object:           Venus
Flyby 2 Date:             Fri January 18th, 2008 13:07:15
Flyby 2 Julian Date:      2454484.5467
Flyby 2 DeltaV:           0 km/s
Flyby 2 C3:               0 km2/s2
Flyby 2 TOF:              127.098 days

DSM 3 Date:               Sat April 05th, 2008 18:43:28
DSM 3 Julian Date:        2454709.2417
DSM 3 DeltaV:             0 km/s
DSM 3 C3:                 0 km2/s2
DSM 3 TOF:                78.2335 days

Flyby 3 Object:           Venus
Flyby 3 Date:             Sat August 30th, 2008 05:48:04
Flyby 3 Julian Date:      2454709.2417
Flyby 3 DeltaV:           0 km/s
Flyby 3 C3:               0 km2/s2
Flyby 3 TOF:              146.462 days

DSM 4 Date:               Thu December 04th, 2008 23:00:47
DSM 4 Julian Date:        2454919.4932
DSM 4 DeltaV:             0.926279 km/s
DSM 4 C3:                 0.857993 km2/s2
DSM 4 TOF:                96.7172 days

Flyby 4 Object:           Mercury
Flyby 4 Date:             Sat March 28th, 2009 11:50:16
Flyby 4 Julian Date:      2454919.4932
Flyby 4 DeltaV: 0 km/s  
Flyby 4 C3: 0 km²/s²  
Flyby 4 TOF: 113.534 days  

DSM 5 Date: Mon November 30th, 2009 15:13:30  
DSM 5 Julian Date: 2455270.6457  
DSM 5 DeltaV: 1.01654 km/s  
DSM 5 C3: 1.03336 km²/s²  
DSM 5 TOF: 247.141 days  

Flyby 4 Object: Mercury  
Flyby 4 Date: Sun March 14th, 2010 15:29:44  
Flyby 4 Julian Date: 2455270.6457  
Flyby 5 DeltaV: 0 km/s  
Flyby 5 C3: 0 km²/s²  
Flyby 5 TOF: 104.011 days  

DSM 6 Date: Wed June 30th, 2010 01:14:37  
DSM 6 Julian Date: 2455423.2462  
DSM 6 DeltaV: 1.07041 km/s  
DSM 6 C3: 1.14577 km²/s²  
DSM 6 TOF: 107.406 days  

Arrival Object: Mercury  
Arrival Date: Sat August 14th, 2010 05:54:28  
Arrival Julian Date: 2455423.2462  
Arrival DeltaV: 0.920366 km/s  
Arrival C3: 0.847073 km²/s²  
Arrival TOF: 45.1943 days  

State Vector:  
2275.51, 2.5, 0.0541367, 0.500283, 0.451116, 492.707, 1.61567, 1.31289, 0.0410242, 170.965, 1.6755, 2.80187, 0.380845, 224.695,  
1.05, 1.57085, 0.458628, 210.335, 1.05, 2.13024, 0.704363, 351.127, 1.05, 1.69513, 0.704958, 152.509  

Mars Round-Trip  
Mission overview and breakdown:  

Mission Overview  
---  
Sequence: EEME  
Departure Date: Wed July 13th, 2016 13:12:45  
Arrival Date: Tue October 05th, 2021 23:53:24  
Total C3: 39.7717 km²/s²  
Total DeltaV: 6.30648 km/s  
Total TOF: 1910.44 days  

Mission Breakdown
Departure Object: Earth
Departure Date: Wed July 13th, 2016 13:12:45
Departure Julian Date: 2457583.5505
Departure DeltaV: 1 km/s
Departure C3: 1 km2/s2

DSM 1 Date: Wed January 11th, 2017 22:53:52
DSM 1 Julian Date: 2458050.3843
DSM 1 DeltaV: 0.707693 km/s
DSM 1 C3: 0.500829 km2/s2
DSM 1 TOF: 182.404 days

Flyby 1 Object: Earth
Flyby 1 Date: Mon October 23rd, 2017 09:13:27
Flyby 1 Julian Date: 2458050.3843
Flyby 1 DeltaV: 0 km/s
Flyby 1 C3: 0 km2/s2
Flyby 1 TOF: 284.43 days

DSM 2 Date: Sat November 10th, 2018 23:53:59
DSM 2 Julian Date: 2458889.6628
DSM 2 DeltaV: 2.28319e-010 km/s
DSM 2 C3: 5.21297e-020 km2/s2
DSM 2 TOF: 383.611 days

Flyby 2 Object: Mars
Flyby 2 Date: Sun February 09th, 2020 15:54:30
Flyby 2 Julian Date: 2458889.6628
Flyby 2 DeltaV: 1.56756 km/s
Flyby 2 C3: 2.45725 km2/s2
Flyby 2 TOF: 455.667 days
Flyby 2 Surface Stay Orbit:
  Semimajor Axis: 37000 km
  Eccentricity: 0.9
  Tstay: 50.019 days

DSM 3 Date: Fri October 30th, 2020 07:36:52
DSM 3 Julian Date: 2459493.9954
DSM 3 DeltaV: 2.2917 km/s
DSM 3 C3: 5.25187 km2/s2
DSM 3 TOF: 263.654 days

Arrival Object: Earth
Arrival Date: Tue October 05th, 2021 23:53:24
Arrival Julian Date: 2459493.9954
Arrival DeltaV: 0.739531 km/s
Arrival C3: 0.546906 km2/s2
Arrival TOF: 340.678 days
References


145