AUTOMATED DETECTION OF INVARIANT MANIFOLD INTERSECTIONS USING GRID BASED APPROACH

BY

JOSHUA AURICH

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Professor Victoria Coverstone

ABSTRACT

When designing spacecraft trajectories, there exist cases where a trade-off between time-of-flight and fuel become crucial to the mission design scenario, especially for unmanned missions where longer time-of-flight solutions can be considered. One effective way to produce longer time of flight solutions is to leverage the natural dynamics of the system, which lends towards low energy trajectories. The dynamical structures of such systems provide global transport in multi-body regimes, and therefore avenues to low-cost solutions in a minimum fuel or Δv sense. Trajectories using the natural dynamics are termed low-energy (LE), and typically include either impulsive or low-thrust control to navigate from one global transport to the next. The multi-body model studied in this thesis is the circular restricted three-body problem (CR3BP) and the dynamical structure of interest are the invariant manifolds of the Euler-Lagrange points. The construction of LE trajectories in the CR3BP is most often accomplished by manually finding homoclinic and/or heteroclinic intersections of invariant manifolds located on specific Poincaré surfaces of section. Historically, these patch-points are chosen by hand and used to seed either differential correction, at most yielding a feasible solution, or a control transcription with nonlinear programming to hopefully yield a locally optimal solution. Manual selection of these patch-points is a severe limitation of the current LE trajectory optimization approach and greatly reduces the chance to identify a globally optimal solution. The focus of this thesis is to present an automated solution which removes the bottleneck of characterization and analysis of these intersections of invariant manifolds. This thesis will demonstrate the application of the functionality to autonomously detect and characterize intersections of invariant manifolds, as well as explore the effects of different parameters on performance and generated solutions.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to first and foremost acknowledge my advisor, Professor Victoria Coverstone, who graciously accepted me into her research group, who helped to guide and mentor me through my graduate school experience, and ultimately provided the environment through which this work could be accomplished. I would also like to acknowledge the work of Vishwa Shah, for creating the global optimization framework through which my tool could even be applied. Without his work and dedication, none of this would have been possible. And last but not least, I would like to thank Ryne Beeson for his advisory role through the development of both my automated manifold intersection identification algorithm as well as the global optimization framework at large.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PCR3BP	Planar Circular Restricted Three-Body Problem
LE	Low Energy
NSGA-II	Non-dominated Sorting Genetic Algorithm-II
NLP	Nonlinear Program
GA	Genetic Algorithm
HOC	Hybrid Optimal Control
CUDA	Compute Unified Device Architecture
MBH	Monotonic Basin Hopping
LCS	Lagrangian Coherence Structures

LIST OF SYMBOLS

Δv	A change in velocity.
L_i	The i^{th} Lagrange point.
$\overline{U}$	The boundary and internal area of a Poincare section.
C	Jacobi Integral.
$\mathcal{W}_i$	Manifold of the L_i Euler-Lagrange point.
x,y	Nondimensionalized position coordinates.
$\dot{x}$,$\dot{y}$	Nondimensionalized velocities.
P_i	The i^{th} point in the $[y,\dot{y}]$ or $[x,\dot{x}]$ plane.

CHAPTER 1

INTRODUCTION

Currently, there exists a need for an automated global optimization toolset for spacecraft trajectory optimization in multi-body problems, where current methodologies are hindered by a demand for human time and intuition. Such a toolset has been in development at the University of Illinois at Urbana-Champaign in the Aerospace Engineering department, and has been applied to a variety of trajectory cases. [1, 2, 3, 4, 5, 6] This toolset is primarily driven by a multi-level optimization framework, which has been spearheaded by Vishwa Shah. [1, 4] To allow for efficient low energy transfers, an algorithm was produced to allow for the capability to detect and classify intersections of invariant manifolds. Through this thesis, an algorithm for the detection and classification of the intersections of invariant manifolds will be presented, variables relevant to this algorithm will be explored, computational time considerations will be analyzed and improved, and example trajectory cases utilizing this algorithm will be demonstrated.

In order to apply the algorithm presented in this thesis, it was necessary to encompass the functionality of being able to explore the intersections of invariant manifolds into a larger global optimization framework, which allows for the exploration of specific mission profiles. When utilizing the presented algorithm within the framework, we are specifically interested in leveraging the natural dynamics of the problem, since this structure provides global transport in multi-body regimes, and therefore avenues to low-cost solutions in a minimum fuel or Δv sense. Trajectories using the natural dynamics are termed low-energy (LE), and typically include either impulsive or low-thrust control to navigate from one global transport to the next. The multi-body model considered in this thesis is the planar circular restricted three-body problem (PCR3BP) and the dynamical structure of interest are the invariant manifolds of the Euler-Lagrange points. The construction of LE trajectories in the PCR3BP is most often accomplished by manually finding homoclinic

and/or heteroclinic intersections of invariant manifolds located on specific Poincaré surfaces of section. Historically, these patch-points are chosen by hand and used to seed either differential correction, at most yielding a feasible solution, or a control transcription with nonlinear programming to hopefully yield a locally optimal solution. Manual selection of these patch-points is a severe limitation of the current LE trajectory optimization approach and greatly reduces the chance to identify a globally optimal solution. The focus of this work is to present an approach for removing the manual selection process. The algorithm presented is an efficient and intelligent automated search process and is able to automatically seed our global optimizers based on ideal sets of points and therefore perform a rapid global search.

Automated detection of dynamical connections is not new and has been demonstrated by Dellnitz et al.[7] and Zanzottera et al.[8] The paper by Dellnitz worked with the planar concentric four body model and used Earth to Venus transfers as an example. A primary focus of their implementation was to produce a data set of all the possible states that could be reached in finite time with approximate control applied to initial states near a prescribed manifold; a reachable set. The initial and final states then prescribe a map. An initial guess to the solution of a continuous closed trajectory can then be found by searching for two maps that have similar boundary conditions (e.g. two final conditions with similar position and velocity states). Although the authors did not close the trajectories in the first work mentioned.

To differentiate the work presented in this thesis, we are interested in LE trajectories using impulsive control and therefore the presented approach looks for intersections of reachable sets produced by way of the natural dynamics. The results of this thesis focus on automated and computationally efficient ways of finding and categorizing these sets of intersections. The results of this thesis are meant to work within a global optimization framework and therefore must be very computational efficient. This is in contrast to the results produced by Dellnitz et al., which was not embedded in an optimizer. The time to completion for producing an intersection result using the algorithm presented in this work is significantly faster than the result demonstrated by Dellnitz.

The paper by Zanzottera is focused on spatial intersections of invariant manifolds in the Sun-Earth-Moon regimes. In terms of finding dynamical structure, their paper and this thesis are similar, although the algorithm

presented is currently restricted to the planar case. Their work was facilitated by a dynamical systems tool called Global Analysis of Invariant Objects (GAIO), which is a suite designed to analyze general dynamical systems. In Zanzottera, the intersection of invariant manifolds of halo orbits are found. Categorizing these intersections into subsets is of interest, since special combinations imply advantageous initial and final conditions (e.g. ballistic capture at initial and final states). The algorithm presented in this thesis is aimed at naturally identifying and categorizing these subsets. It is unclear if Zanzottera et. al. carry out this additional step, although GAIO most likely enables this type of analysis. The main difference between this work and Zanzottera is that this algorithm is geared towards not only autonomous identification of intersection set, but also categorization and doing so in an extremely computationally efficient manner.

CHAPTER 2

BACKGROUND

2.1 The Planar Circular Restricted Three-Body Problem

The algorithm and results presented in this thesis use the planar circular restricted three-body model. For this work, a massless body represents the spacecraft, whose small mass has no meaningful effect on the orbits of any parent body. The two massive bodies are a primary body and a secondary body, for example the Sun and Earth respectively. These bodies orbit around their barycenter and affect the motion of the massless body. Both bodies are assumed to have zero eccentricity and inclination. The reference frame for this problem is centered at the barycenter. Additionally, a synodic (rotating) reference frame is used which is given the same angular velocity as that which the secondary body rotates about the primary body, which is scaled to be $\omega = 1$. In this way the bodies in this reference frame are visually seen to be fixed along the x axis. Given a mass of the primary body to be m_1 and a mass of the secondary body to be m_2, the value μ is defined as

$$\mu \equiv \frac{m_2}{m_1 + m_2}$$

In terms of μ, the states of the primary and secondary in the canonical frame are: $(-\mu, 0, 0, 0)$ and $(1-\mu, 0, 0, 0)$. In the canonical frame, the equations of

[1]This chapter contains previously published material from [2] and [5]. The copyright owner provides permission to reprint.

motion for the massless third body are,

$$\ddot{x} = \quad 2\dot{y} + x - (1-\mu)\frac{x+\mu}{r_1^3} - \mu\frac{x-1+\mu}{r_2^3}$$

$$\ddot{y} = -2\dot{x} + y - (1-\mu)\frac{y}{r_1^3} - \mu\frac{y}{r_2^3}$$

where $r_1 \equiv \sqrt{(x+\mu)^2+y^2}$ and $r_2 \equiv \sqrt{(x-1+\mu)^2+y^2}$ are the distances of the massless body from the primary and secondary respectively. Setting the velocity and acceleration terms to zero yields five stationary solutions that are called Euler-Lagrange points (see Koon et. al.[9] or Parker[10]). The locations of these solutions are shown in Figure 2.1. For most applications, including the systems consider in this thesis, the L_1, L_2 and L_3 points are unstable fixed points, whereas L_4 and L_5 are stable. For large values of μ, the L_4 and L_5 undergo a bifurcation and also become unstable.

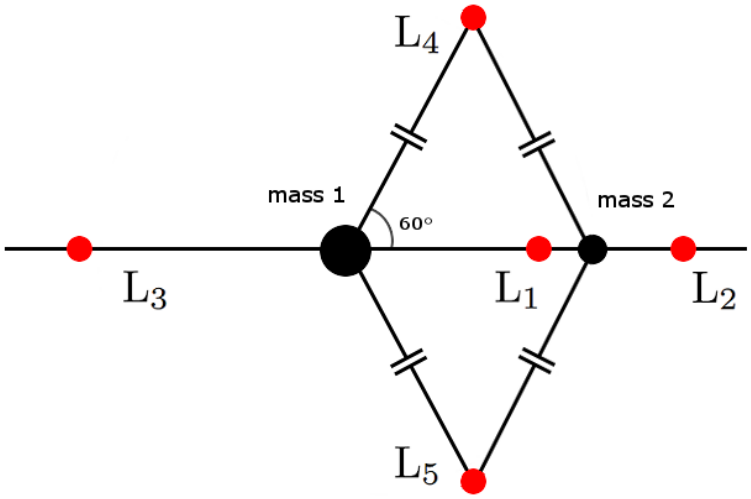

Figure 2.1: Diagram of the PCR3BP, with 5 Euler-Lagrange points identified.

The co-linear Euler-Lagrange points, L_1, L_2 and L_3, are of special interest, since their instability gives way to unstable periodic orbits about each point. In the remainder of this thesis we will focus on the L_1 and L_2 points. Being a Hamiltonian system, we have an integral of the motion, which provides a method for dynamical reduction. It is common to work with what is often

termed the *Energy Integral*, given as follows

$$E \equiv \frac{1}{2}(\dot{x}^2 + \dot{y}^2) - \frac{1}{2}(x^2 + y^2) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2} - \frac{1}{2}(1-\mu)\mu$$

In this thesis, we will work with the equivalent *Jacobi Integral*, given as $C \equiv -2E$. It is useful to separate E into kinetic and effective potential terms, with $E = V + \overline{U}$ and V and $\overline{U}$ defined as,

$$\overline{U} \equiv -\frac{1}{2}(x^2 + y^2) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2} - \frac{1}{2}(1-\mu)\mu$$
$$V \equiv \frac{1}{2}(\dot{x}^2 + \dot{y}^2)$$

One important feature to note is the ordering in terms of energy of the three co-linear Euler-Lagrange points; which is the following: $C_1 > C_2 > C_3$. C is used to perform dynamical reduction and thus enable comparison of heteroclinic and homoclinic connections on reduced state spaces.

2.2 Lyapunov Orbits and Invariant Manifolds

Invariant manifolds are dynamical structures which exist about periodic orbits in the PCR3BP and are generated within global optimization framework to be used as dynamical structures to detect low energy trajectories. About each periodic orbit, there exist both stable and unstable manifolds which flow both into and out of the periodic orbit respectively.[11, 9] For use in this work, invariant manifolds are produced by perturbing the spacecraft away from halo or Lyapunov orbits based upon stability information from the state transition matrix, and capture how the flow of the spacecraft propagates over time. In the case of the stable manifold, the flow will tend towards the original periodic orbit. Conversely, the flow of the unstable manifold leaves the periodic orbit when perturbed. These flows caused by perturbations are captured through the propagation of individual arcs using the three-body dynamics, whereby the superposition of a variable number of discrete arcs are necessary to define the boundary region of any given manifold.

While these structures generally tend towards a tube-like shape as they grow away or towards a periodic orbit, the dynamics of the problem can generate complex flows. These systems are chaotic, meaning among a variety

of characteristics that arise when dealing with chaotic systems, that they are heavily dependant on initial conditions and can diverge significantly over time. When looking at a visual representation of a manifold, it can be the case that some portion of the arcs continue on a visually predictable path, while other arcs break away and flow along wildly varying paths. Many times this variation can be caused by close approach to one of the planetary bodies in the PCR3BP, where some arcs can become ejected. The formation of a tube-like structure by stable (green) and unstable (red) invariant manifolds can be seen below in Figure 2.2.

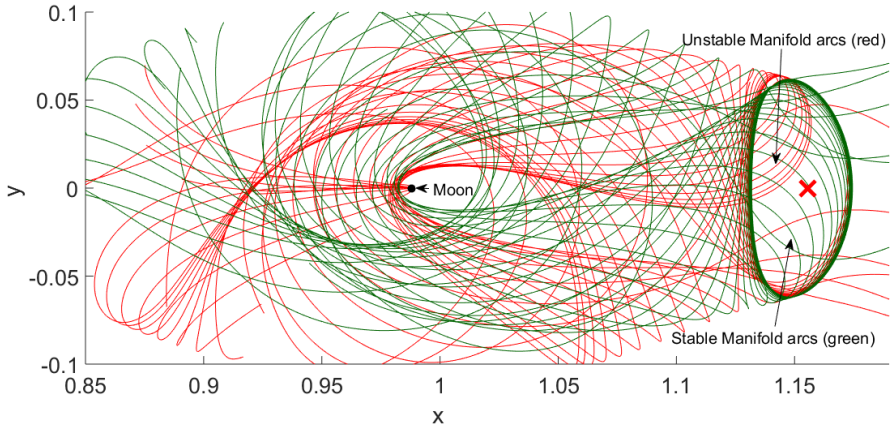

Figure 2.2: The propagation of arcs along the unstable (red) and stable (green) Earth-Moon L_2 manifolds. [1]

2.3 Poincaré Surfaces of Sections

A Poincaré surface of section is a useful tool for dynamical reduction and thus enables analysis on a reduced space. The combination of the PCR3BP being an autonomous Hamiltonian system coupled with the use of Poincaré surfaces of section allows an individual to 'see' the full complexity of the dynamics of the PCR3BP. Poincaré surfaces of section can be defined quite easily, with the only requirement being that the local flow be transversal to the surface of section; this is often slightly abused in practice. We will designate Poincaré surfaces of section as W_i where i serves as a label. For instance see Figure 2.3.

Poincaré surfaces of section are used in this thesis as designated planes

where we record the intersection of invariant manifolds. Recording these intersection points of various invariant manifolds and using dimensional reduction via constant C, we can search for states that belong to intersecting manifolds (i.e. homoclinic and heteroclinic intersections). The remainder of this work will detail methods and an algorithm for efficiently carrying out this process in an automated fashion. It should be noted that for brevity, we will loosely refer to the subsets of Poincaré surfaces of section, which correspond to intersections of invariant manifolds, simply as Poincaré sections themselves. This terminology avoids restating that we are actually looking for subsets of intersections of Poincaré surfaces of section.

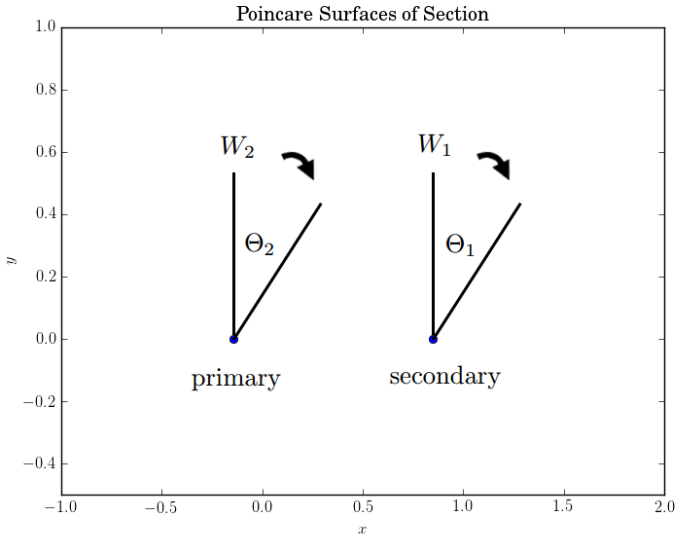

Figure 2.3: Example of locations of Poincaré surfaces of section when they are located at either the primary (W_2) or secondary (W_1) in the PCR3BP. A common Θ_1 is 270° and a common Θ_2 is 0°.

An example Poincaré section and its generation is described. By starting at the Sun-Earth L_2 Lyapunov orbit with a Jacobi interal $C = 3.0008$, the resulting manifold $\mathcal{W}_2^{U,i}$ is propagated until it intersects with the surface. The surface is located at the Earth in the Sun-Earth synodic frame where $y > 0$. The generated Poincaré section can be seen in Figure 2.4.

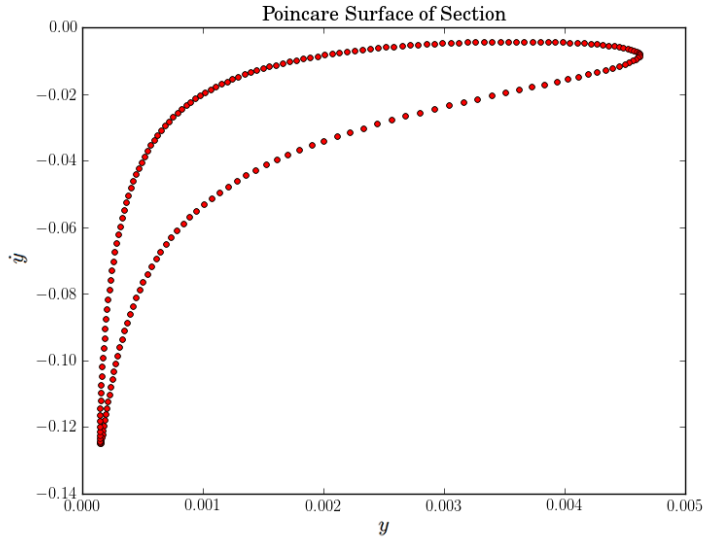

Figure 2.4: 200 arcs are propagated along the Sun-Earth L_2 unstable manifold $\mathcal{W}_2^{U,i}$, and their intersection with the Poincaré surface defines a bounded region.

2.4 The Hiten Mission

The Hiten mission was executed by the Japanese space agency in 1991. This mission used a LE transfer with a ballistic capture at the Moon. One side effect of the LE approach to this mission was that the total Δv requirement to complete the transfer was actually less than a standard Hohmann transfer. The construction and solution of Hiten using a patched three-body approach is clearly articulated by Koon et al.[12]. To explore the algorithm presented in this work, generating solutions of a Hiten-like mission would demonstrate several desirable capabilities: the first being the ability to automatically detect intersections of invariant manifolds from different three-body systems due to the need for a patched three-body approach, and second being the automated generation of closed impulsive trajectories using boundary conditions from these intersections, hence applying the algorithm. The goal of this work is not to re-solve the actual Hiten mission, but instead solve a variant of what is being termed a Hiten-like trajectory that allows investigation of the capabilities of the algorithm presented in this thesis.

In this work, we simplify the Hiten mission by targeting a fixed circular parking orbit at the moon instead of a ballistic capture. This means that the solutions presented in this thesis are not expected to demonstrate the

same low Δv values as seen in Hiten, but instead should simply demonstrate the same qualitative concepts while demonstrating the capabilities of the automated detection of invariant manifolds. Future efforts could extend this analysis on the true Hiten mission.

The Hiten-like trajectory proposed in this thesis is characterized by a few distinct steps. First, the spacecraft starts at an Earth parking orbit. An initial burn is executed so that a solution either travels along or shadows a Sun-Earth L_2 unstable manifold. A second burn is then used to transfer onto an Earth-Moon L_2 stable manifold. The final step is traditionally a ballistic capture at the Moon, but in this work the spacecraft is inserted into a lunar parking orbit, which is circular with 1,000km altitude. For the initial Earth parking orbit, the spacecraft will consider initial circular orbits with ranges between 800-10,000km.

CHAPTER 3

ARCHITECTURE

3.1 Global Optimization Framework

As a primary methodology to apply the presented automated invariant manifold intersection detection capability, an optimization framework was employed which can effectively search parameters of interest on a per mission basis. This framework has been under development at the University of Illinois at Urbana-Champaign, which uses a genetic algorithms, coupled with a NLP solver for local optimization, in a hybrid optimal control (HOC) framework to optimize LE transfers[4, 1]. In this section, a basic description of how this optimization framework operates is presented and shows how this tool can explore large search spaces to determine optimal solutions to LE trajectory problems.

The optimization framework employs a two level optimization structure. Figure 3.1 provides a depiction of the framework. At the outer loop level, high-level control parameters are selected via a genetic algorithm (GA). These values are then used to construct local optimization problems at the inner loop level. It is at the inner loop that a nonlinear programming (NLP) problem is formed and solved with the NLP solver, which in the case of this work was chosen to be SNOPT. The objective values for the resulting candidate are then evaluated (e.g. time-of-flight, Δv, or propellant mass used for the low thrust case) where the GA ranks the results, and produces a new generation of scenarios. Ideally, this process should emulate evolution in nature, where over sufficient generations more desirable solutions should pass on their genes to produce candidate trajectories with favorable qualities.

The selected optimization framework uses the Non-dominated Sorting Ge-

[1]This chapter contains previously published material from [2] and [5]. The copyright owner provides permission to reprint.

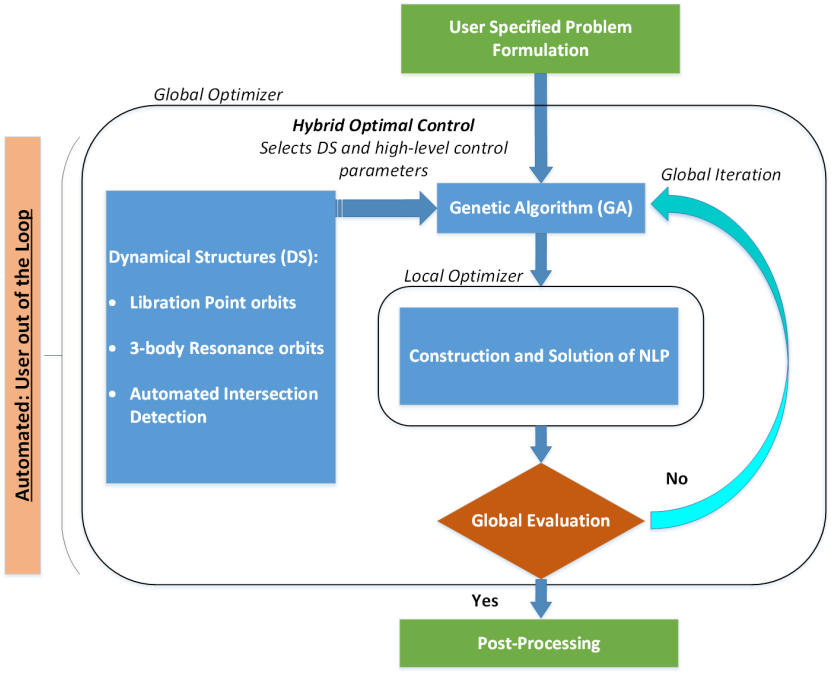

Figure 3.1: The flow of the automated global optimization framework.

netic Algorithm-II (NSGA-II) solver in its outer loop, which enables the solution of multi-objective problems. Many GAs are only single objective which would not be sufficient for this application. This algorithm starts with a set number of individuals, which make up a population. The effective genes which make up an individual are randomly selected from a prescribed range on a per variable basis. After the initial population is set up, this population is evolved over some number of generations. As the generations progress, the solutions should improve with respect to any of the specified objective values. In this work I am only considering two objective problems. At each generation, there are three processes which take place on the population: namely crossover, mutation, and selection. Crossover is performed between two parent individuals, where combinations of their genes are used to produce two new children individuals. Mutation is performed on some number of individuals per generation to mix up some randomly selected genes to allow for greater diversity in the population. And lastly, selection is performed between the old and new populations, maintaining elitism, to select a batch of the best individuals to move forward in the final version of the population for a given generation.

Within the inner loop, trajectory optimization becomes the primary focus.

This is accomplished by transcription of an optimal control problem into a NLP where a solution is generated using SNOPT. The NLP is a general problem that allows for bounds on the control parameters, nonlinear constraint functions and nonlinear objective functions.

Lastly, since our problem has multiple objectives, one needs to define what it means for one solution to be "better" than another. We use the definitions of non-dominated solutions and Pareto fronts to resolve this issue. Plotting Pareto fronts then provides a concise way to understand how one can trade the objectives in one solution with respect to another. In our case, this will be a two dimensional plot of Δv versus time of flight. For further details regarding the implementation of the optimization framework, see Shah et.al.[4, 1]

3.2 Utility of Automated Invariant Manifold Intersection Detection

The three-body problem is an extremely useful model which can be chained to form approximations to larger systems (e.g. the Jovian system). It is a chaotic system with natural dynamics that can be leveraged for best seeding the inner-loop and it is this problem of strategically seeding the inner-loop that this work can be applied. Specifically, results in this thesis will look at how to efficiently generate new initial guesses for the boundary conditions of control phases in the three-body problem. Since we are interested in LE solutions, this is carried out by finding the intersection of heteroclinic and homoclinic connections on specific Poincaré surfaces of section. The invariant manifolds of these connections asymptotically tend toward unstable periodic orbits at the Euler-Lagrange points forward (stable) or backward (unstable) in time; details are given in Section 2.1. Generating these initial guesses occurs at the outer-loop level, which may be controlled by a number of global optimization searches; including algorithms for multi-objective optimization, which is the focus of Shah et. al.[1]. The efficient generation of initial conditions for LE trajectories with boundary conditions on 'free' manifold arcs (i.e. those that are not meant to patch two separate LE manifold arcs at a given Poincaré surface of sections) are detailed in Beeson et. al.[3]

3.2.1 Manifold Sections

By intersecting manifold structures with a plane, a Poincaré surface of section can be produced to describe the flow of a spacecraft. Since any particular orbit around a body may take a spacecraft through a given surface multiple times, sections can be further divided into passes which are differentiated by the count of intersections with said surface. For the purposes of this algorithm, there are two cases for any set of points produced on a Poincaré surface of section within the PCR3BP. In the first case, the points join to produce some closed and bounded shape, which has some definite interior area. The other case, which is fairly common especially for multiple passes or high energies, is that some of the arcs belonging to the manifold diverge from a tube-like structure. One common way this happens is if some arcs pass closely to either the primary or the secondary. This creates a significant range of velocities as the arcs are ejected from the system in various directions. The end result is that the arcs produce a set of points on the Poincaré surface which do not bound any internal area. An example of a closed and bounded set compared to a poorly defined set on a Poincaré surface can be seen in Figure 3.2.

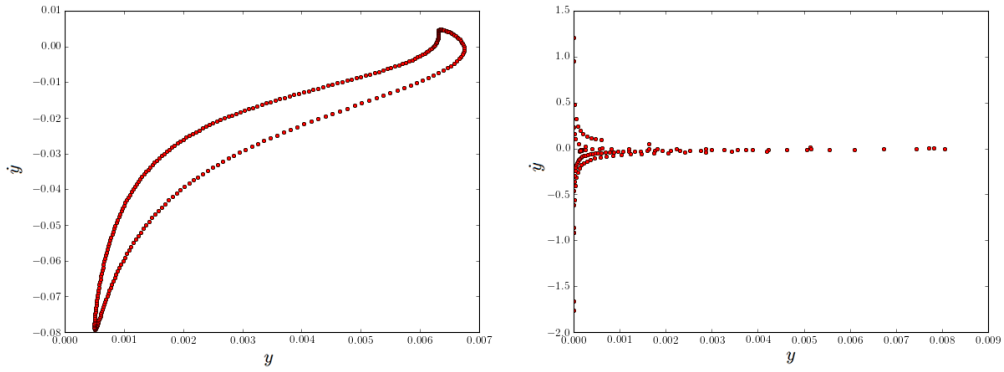

Figure 3.2: On the left, a closed and bounded ellipsoidal shape is produced by intersecting the Sun-Earth $\mathcal{W}_2^{S,i}$ manifold with a Poincaré surface of section. On the right, the same manifold with a higher Jacobi integral produces a curvature with poorly defined internal area on the same surface. The well bounded set was produced with a Jacobi integral of $C = 3.0008$ and the poorly bounded set was produced with a Jacobi integral of $C = 3.0007$.

From a functional point of view, the main differences between the two sets of points seen in Figure 3.2 are a difference of internal area and variation from

an ellipsoidal structure. Even for cases which are bounded and closed, the sets of intersection points found on the surface may loop back on themselves and lose all resemblance to an ellipsoidal shape. The algorithm developed has been designed to identify these differences from an open ended perspective. Namely, the algorithm does not make assumptions whether the set of points captures any internal area or not, nor apriori the shape for which we are attempting to identify.

To elaborate on the algorithm, let us define ∂U_j^i to be the boundary region of the i^{th} intersection with the Poincaré surface of the j^{th} manifold and let U_j^i be the interior bounded region, which may be an empty set. Additionally define $\overline{U}_j^i = \partial U_j^i \cup U_j^i$. By looking at two overlapped Poincaré surfaces $\overline{U}_1$ (Green) and $\overline{U}_2$ (Red), each with these two identified subset regions, we can develop a variety of additional subsets which can be used to seed our optimizers.

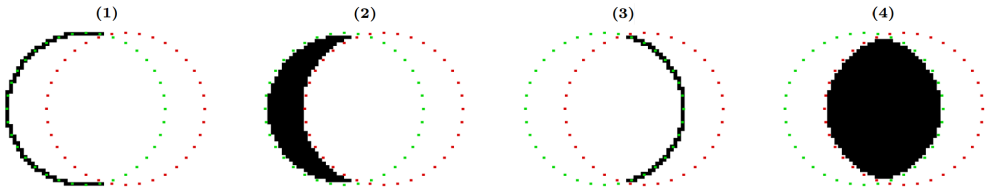

Figure 3.3: From left to right: (1) ∂U_1 Boundary external to $\overline{U}_2$: $\partial U_1 \setminus \overline{U}_2$, (2) U_1 Internal external to $\overline{U}_2$: $U_1 \setminus \overline{U}_2$, (3) ∂U_1 Boundary Inside U_2: $\partial U_1 \cap U_2$, (4) Shared Internal: $U_1 \cap U_2$.

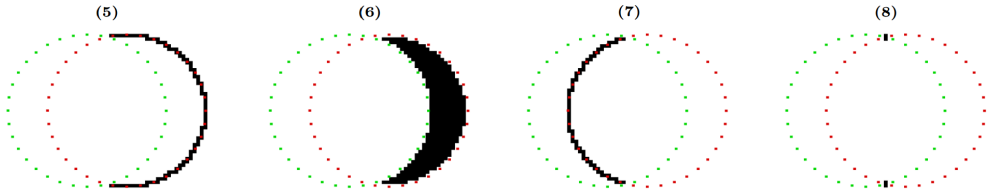

Figure 3.4: From left to right: (5) ∂U_2 Boundary external to $\overline{U}_1$: $\partial U_2 \setminus \overline{U}_1$, (6) U_2 Internal external to $\overline{U}_1$: $U_2 \setminus \overline{U}_1$, (7) ∂U_2 Boundary Inside U_1: $\partial U_2 \cap U_1$, (8) Intersection: $\partial U_1 \cap \partial U_2$.

Two areas to be identified are found by looking at the unique parts of each Poincaré section, producing (1) and (5) for the boundary and (2) and (6) for internal area. The remaining regions are shared by both sets. An especially

useful region is where the boundaries of both sets intersect, which produces (8). Additionally of interest is where the internal areas of both sets overlap, which produces (4). The remaining two regions are where the boundaries of one set intersect with the internal areas of another, producing (3) and (7).

3.2.2 Utility of Sections

While these subdivisions of the Poincaré sections may seem like simple applications of set theory to bounded regions, there exists a deeper understanding and useful application which can be extracted by applying the lens of dynamics to this problem. By numerically exploring the spaces inside these regions, interesting characteristics may be found within the dynamics of the problem which can be used to achieve specific objectives. To understand how the dynamics of the problem can be applied, it helps to first reiterate how these sets of points were produced. By starting at some Lagrange point in the PCR3BP, a series of trajectories were perturbed away along directions dictated by stability information from the state transition matrix and propagated through space until they intersected with a prescribed Poincaré surface of section. In this particular application, this is done with two separate manifolds and/or energies and the results overlaid onto the same surface, sometimes after the application of a coordinate frame change. These points on the Poincaré surface therefore represent a snapshot of where a spacecraft would have been had it been traveling with those given states, and had it continued to propagate would have either traveled away from or towards the periodic orbit, depending on if we are discussing a unstable or stable manifold respectively. Therefore, if our states were to exist anywhere on the boundary region of a particular manifold's Poincaré section, see Figures 3.3 and 3.4, the spacecraft would maintain the behavior of that manifold. For example if the Poincaré section was produced from a stable manifold, had the spacecraft exist anywhere on the boundary, it would flow back onto the periodic orbit. If the states were to exist inside the internal region of our Poincaré section, the spacecraft would approach and fly through the periodic orbit, but not capture onto the periodic orbit. Where the utility of applying an understanding of the dynamics becomes particularly useful is when taking into account the combination of two overlapping Poincaré sections. If a

state inside the shared internal area of both Poincaré sections is chosen, the spacecraft would then propagate through both periodic orbits, but capture on neither of them. If a state on the boundary of the first Poincaré section inside of the internal area of the second section is chosen, then one would see that the spacecraft propagates through the second periodic orbit and captures onto the first periodic orbit. It is with these expected behaviors that mission objectives can be numerically explored using these regions.

3.3 Automated Poincaré Intersection Algorithm

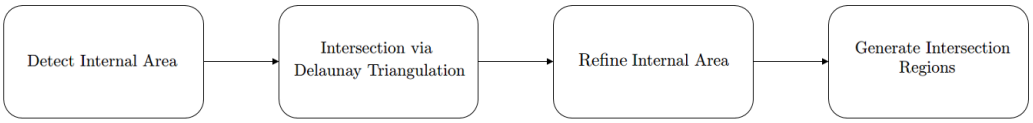

Figure 3.5: Basic algorithm steps.

In this section, the flow of the algorithm and intricacies which have been considered during development are presented. The overall objective to be explained is how to take two sets of points generated by intersecting manifolds with a Poincaré surface of section, and through a series of computations, produce defined sets or regions, namely (1)-(8) from section 3.2.1. While each set is defined by simple set notation, the process of identifying the boundary and internal regions of each Poincaré section independently can be challenging, as well as handling the many edge cases which arise through the interesting variety produced through the dynamics of the PCR3BP. Therefore the main steps in generating the final result are presented in four segments, see Figure 3.5, for reference purposes.

Detecting Internal Area

The flow of the algorithm starts first with each Poincaré section separately, and only combines them after sufficient definition of each section has been established. A Poincaré section supplied to this algorithm has no real constraints on the size or shape of the points being provided. One such edge case arises where it could be that while many arcs were perturbed from the original periodic orbit, only a few or maybe none at all intersect with the

Poincaré surface. From this perspective, since the set of these points are used to define the boundary of some shape, a minimum number of intersections with the surface for each Poincaré section is imposed, where if either Poincaré section has less than this amount, the algorithm does not even try to define the regions. For this work, this value was selectively chosen to be 20 points. While typically 100-300 arcs are perturbed per periodic orbit in an effort to define a quality boundary region, the case of 20 or less intersection points happens very rarely. Such a low value for number of intersections when propagating more than 100 arcs is usually reserved for higher energy manifolds which can be expected when the genetic algorithm is exploring a variety of energy options.

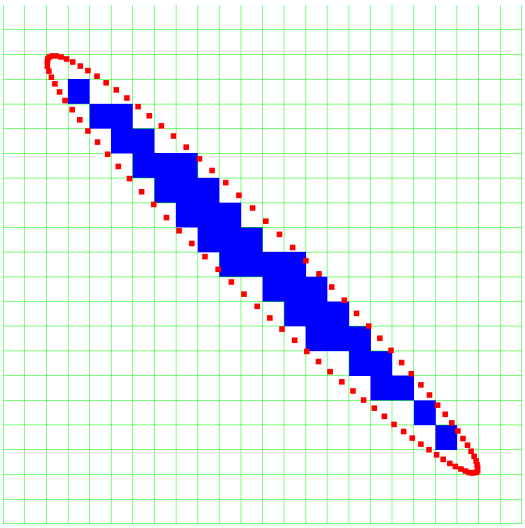

Figure 3.6: Coarse internal area found through adaptively sizing a grid.

If a sufficient number of intersection points are supplied for each Poincaré surface, an adaptively sized grid is produced based upon the bounds of the supplied points. This grid fits around the intersection points with empty buffer space on all sides. This buffer space is added because further in the algorithm it is assumed that the exterior of the grid will be void of intersection points. The dimensions of the grid squares are chosen based off the distances between the intersection points. Namely, a list of unique distances sorted from largest to smallest based on distance from each point to its nearest neighbor is constructed. One useful characteristic of this approach is that it is guaranteed to be close to the expected grid sizes necessary to detect internal

area. The routine begins with the largest distances which would produce the smallest in number and largest sized grid squares, with progressively higher resolution grids trialed in the future if the exit condition is not satisfied. This process is repeated until some internal area is found, where the boundary region ∂U_i is roughly defined by the region of grid squares which contain at least one intersection point. At this stage, the characteristic that the exterior region of the grid being void of points comes into use. By starting in any corner, grid squares can easily be sorted into three categories. First, there is the group of all empty grid squares, which are found by finding all empty squares connected to a corner square, which is guaranteed to be empty. Then the group which includes all grid squares which contain at least one intersection point is identified. Once these two groups are defined, the remaining unassigned grid squares are deemed to be some roughly defined internal area U_i and the routine is successful in finding internal area for this set of points. Since some internal area has been found, it can be assumed that this set of points approximates a closed and bounded set. However, this defined internal region and boundary region are very coarse, and are only used in the future to seed additional passes at a higher fidelity solution. An example of a coarse internal area estimated output can be seen in Figure 3.6. If this routine is successful for both of the provided Poincaré sections, then the algorithm can delve deeper into defining the targeted regions from section 3.2.1.

Intersections via Delaunay Triangulation

Now that each Poincaré section has been analyzed individually, for there to exist an intersection, the meaningful contents of one Poincaré section must overlap with that of the other. While a grid based method could also be used for this purpose, it is expensive in the sense of time and memory to produce another grid prematurely which combines the two sections, just to find that they do not overlap. To avoid this potentially expensive cost, a computationally efficient method of detecting intersection is implemented by producing a Delaunay triangulation for each Poincaré section and then intersecting the two triangulations. This method is both computationally and memory efficient and scales in complexity based off the number of intersection points, rather than the size of the respective grids of the two Poincaré

sections, which could produce many grid squares to be checked. One final benefit of this algorithm is scalability. The process of intersecting triangles is something which is very popular in computer graphics, where devices called graphics processing units, or GPUs, exist which have been designed to perform these operations in a way which is parallelized. While the functionality to perform these intersections on the GPU is not currently implemented, it is a target for future work and will make for an even faster intersection check. This will become especially useful when the need arises for large scale distribution of this algorithm. An example of a set of intersection points after being triangulated can be seen in Figure 3.7.

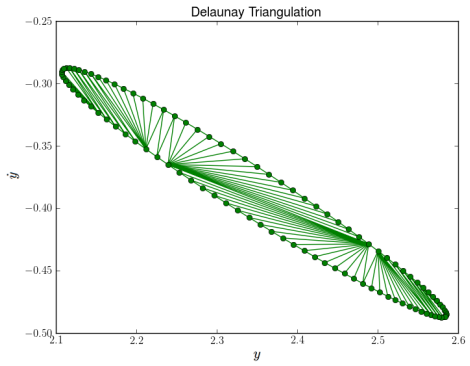

Figure 3.7: An example Delaunay Triangulation from a set of 100 points.)

Refining Internal Area

At this stage of the algorithm, a coarse definition of the ∂U_i and U_i sets for each Poincaré section have been defined, and it has been proven that the individual Poincaré sections intersect. Now a primary goal is towards refining the coarse initial estimate for U_i to better define the regions (2), (4), and (6). This is important since the coarse estimate, an example of which can be seen in Figure 3.6, can miss significant portions of the internal area or include unwanted area from the boundary region ∂U_i. Again, the utility of the adaptively sized grid is employed. Since the original coarse estimates were stopped after any internal area was found, there is very likely room to improve this approximation. This is done by independently reducing the grid sizes in both axis directions until right before the point that the internal and external area become indistinguishable, or namely that a gap is generated in

∂U_i that bridges the empty set and U_i. This gap is detected using the same search implementation that was used to find the empty region for the original coarse estimate, which relies on the edges of the grid being empty. This significantly improves the estimate of the internal area, especially near the boundary region ∂U_i, and analysis of how many steps to take in attempting to optimize the internal area and the effects on performance can be found in Section 4.2.

Generate Intersection Regions

Preparations to produce the remaining sets by combination of both Poincaré sections by this stage have been completed. The next step in generating the output is to appropriately size a new grid which will be used as it is a combination of two different sized grids. In the current implementation, this is chosen to be the smallest of the sizes in both the axis directions from both independent Poincaré sections in order to produce the highest fidelity solution without needing to down-sample. This grid is larger in size and takes longer to search than either of the individual Poincaré section grids due to not only the small sizing, but also the increased area over which the intersection is looking.

The main objective is now to produce the ∂U_1, ∂U_2, and $\partial U_1 \cap \partial U_2$ regions. First, for the set of all points in each Poincaré section, their corresponding locations on the grid are separately populated. While this was sufficient for producing a rough estimate while originally detecting internal area, the accuracy of the boundary regions ∂U_i at this point will be significantly lacking. Due to smaller grid sizes, there may be significant gaps between points which do not now define the desired closed region expected of the boundary. In order to approximate the entire boundary of closed shapes, a routine to trace the boundary region has been implemented. For each set, a point is chosen nearest to the average, or center, of the set to be the starting point P_0, and its nearest neighbor, P_1, is found. For P_1, we then find the nearest neighbor P_2 excluding the starting point, which looking at the dot product between the segments $[0,P_1\text{-}P_0]$ and $[0,P_2\text{-}P_1]$, provides a resulting angle. A variety of P_2 options are checked, until one within an acceptable range of angles is found. A line is drawn between the points P_1 and P_2 on the grid, adding to the boundary region of that Poincaré section all grid squares which intersect

with the generated line. The resulting acceptable P_2 is then selected as the next point on the boundary to be drawn, where $P_0 \leftarrow P_1$ and $P_1 \leftarrow P_2$, and the search for a new P_2 continues. If we cannot find a nearest neighbor which falls within this angle limit, the angle limit is increased by some delta. This walk of [P_0,P_1,P_2] is repeated until the nearest neighbor P_2 that is found is the original starting point again, and the boundary region is closed. When producing these boundary regions for each Poincaré section on the shared grid space, if the boundary regions ever overlap, then the set of intersection of the two boundary regions (8) is populated. While this method generates a boundary region which closely approximates that of the actual boundary region, it was not used originally when finding internal area because using the described routine based on the dot product is prone to artifacts. Sometimes sections can be cut off or line segments unexpectedly traced through the internal area, effectively cutting off entire chunks of the actual set of points. While methods can be employed in the future to reduce these artifacts, currently it can produce false positives for boundary regions, which could potentially produce false positives for internal area, which is highly undesirable. Now that the two boundary regions ∂U_i have been properly identified, namely sets (1), (3), (5), and (7), again the external empty set can be defined by starting in a corner and finding all connected squares which are not determined to belong to sets (1), (5), or (8). Closed boundary regions for each of the Poincaré sections allows for easy identification of internal and external regions, specifically the remainder of regions (2), (4), and (6). Using the internal areas found in the previous portion of the algorithm, the internal areas are overlaid onto the grid, where overlapping of U_1 and U_2 produces $U_1 \cap U_2$, or set (4). The shared internal area is expanded to encompass its entire allowable area, only stopping for boundary regions. The same process is executed for the internal regions of each Poincaré section separately, except only stopping for their own boundary regions. In this way, it is straight forward to identify the remaining subsets.

Edge Cases

At this point there is only one additional topic to consider with respect to the development of this algorithm, and that is the edge cases which inevitably arise. The primary edge case which is captured can be seen in Figure 3.8.

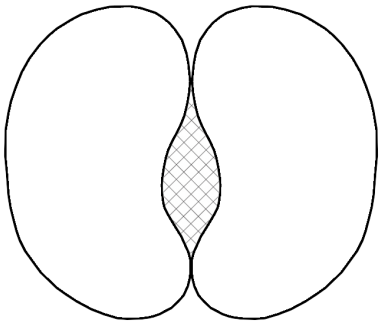

Figure 3.8: Edge case of currently unassigned grid squares represented by hashed area.

In this particular case, there exists some area which has not yet been assigned to any set since the empty external set was not able to reach this region as well as no internal area was assigned either. This case can be easily tackled by masking in both the Poincaré section frames separately, where external regions are assigned unique identifiers. By looking at the overlap of both of these masks at these currently unassigned points, it can be seen that these points belong to neither set $\overline{U}_1$ nor $\overline{U}_2$ and should be included in the empty external set.

One observation which may come to mind is that these regions only approximate the actual boundary and internal regions. This may generate concern if for example one is trying to approach high levels of accuracy in estimation. This particular approach generates more of a fuzzy region around the actual sets which are attempted to be defined. While this may be a legitimate problem for certain uses of Poincaré sections, in our particular application, this level of fidelity is more than sufficient as we only require a small neighborhood to seed initial guesses. By feeding an optimizer with randomly generated estimates from a fuzzy region, the optimizer is capable of slightly adjusting the original estimate to focus in on a preferred solution. By starting in a close enough neighborhood, we give the optimizer a strong chance to find quality results for trajectories.

3.4 Applying CUDA in a Global Optimization Framework

Throughout this thesis, it has been demonstrated that given two sets of points, each representative of the intersection of an invariant manifold with a Poincaré surface of section, intersections of these sets of points and identification of subsets of importance can quickly and reliably be produced[2]. In worst case tests, the time to identification approached 1 second; implying that in a global optimization framework that this process would no longer be a bottleneck. However, it was assumed that timing started once a set of intersection points were provided to the algorithm. Unfortunately it is the case that this process of producing the actual sets of intersection points is actually a major timing concern. In a serial implementation, integrating 350 arcs until intersection with a Poincaré surface of section could take tens of seconds. To compound this problem, this process must be repeated a second time to produce an intersection set for the second manifold case, adding to the total time cost, which now significantly outweighs the potential single second it may take to identify and characterize an intersection. Within this total time cost, there are two primary time intensive tasks being performed. First, the states which are perturbed off the invariant manifold must be generated. By dividing the time along the periodic orbit, in this example, into 350 sections, 350 states are generated at which perturbations can be applied. At this point, each of the states is serially integrated to detect intersection with the Poincaré surface of section. Timing analysis of these two time intensive tasks shows that upwards of 70% of the total time is spent subdividing the periodic orbit, and the remaining time is spent actually integrating the states. This was identified as a necessary target for speed improvements when applying the automated intersection detection of invariant manifolds in a global optimization context where potentially hundreds or thousands of different invariant manifolds of different energies must be checked by the GA. To tackle this problem, use of discrete CUDA-enabled GPU devices was chosen as the target platform for development.

3.4.1 CPU vs. GPU Architecture

The speed improvements seen using the GPU for this particular application are gained due to fundamental architectural differences between the CPU and the GPU. Current consumer grade CPUs have anywhere from two cores for a generic laptop processor to eight cores for higher end devices. If an application has been designed to take advantage of this architecture, for example utilizing multithreading or an MPI architecture to actually employ the capabilities of multiple processors, then speed improvements for parallelized portions of the code can be realized. It has become common, that on many shared cluster environments, users can access up to 100 or more processors. When utilizing the GPU as a resource, the number of cores that can be used becomes significantly larger. For instance, the NVIDIA GTX 570, a GPU which was released in 2010, has a computational capability of 480 CUDA cores. High end graphics cards, such as the NVIDIA GTX 1080 or the NVIDIA Tesla K40 supply the computational capability of 2048 and 2880 CUDA cores respectively. For a single piece of GPU hardware, orders of magnitude more parallelization capability over what can be realized with a high end CPU can be found.

There are some considerations which must be discussed, since it is not fair to directly compare the cores of a CPU and GPU without also discussing clock speed. In general, the clock speed for a CPU can vary between around 1.2 GHz to 3.4 GHz. In comparison, the GTX 570 operates with a clock speed of 732 MHz, while the GTX 1080 and Tesla K40 operate with clock speeds of 1.7 GHz and 745 MHz respectively. In this respect, the CPU dominates. Additionally, when one wishes to mix GPU computations on a process generally running on the CPU, there is a cost associated with sending data, in our case the states to be integrated, to the GPU and executing the kernel. Taking all of these items into consideration, there is a crossover point where the GPU begins to be the preferred computational device, and is generally determined by how parallelizable the problem is, how large each batch of processing is, and balancing the differences in clock speed and employed cores between the CPU and GPU.

3.4.2 Grid and Warp Sizes to Optimize GPU Runtime

GPUs have multiple layers of parallelism within the silicon of the device which are important to take maximum advantage of the device to optimize runtime. When executing a CUDA kernel, which is to say executing some code on a CUDA device, the user may specify a number of grids to be used and a number of threads per grid to be assigned. To optimize across these values can be challenging, and is very much dependant on the code being executed on the CUDA-enabled device. To better understand this operational environment and attempt to maximize performance, an attempt at optimizing the number of threads per grid was performed, with the results shown in Figure 3.9. For this test, a simple scaleable and parallelizable integration scenario was implemented to be executed on the CUDA device. N states are integrated using the equations of motion from the PCR3BP as quickly as possible, with each integration being a standalone operation which will run on its own CUDA core. An $N * 4$ double array containing the N states to be integrated is loaded into memory on the CUDA device. A CUDA kernel which performs the integration routine is executed using a variety of threads per grid sizes, where the total runtime is recorded for each instance. Thread sizes are generally recommended to be powers of 2, therefore thread sizes of $T = 2^n$ where $n \in \mathbb{Z} \cap [0, 9]$ were used. This provides for grid counts $G = \frac{N}{2^n} + 1$. With these values defined, it can be seen that across integration amounts, the optimal thread count per grid varied. The actual operation of how the GPU divides the work load and optimizes its execution time per thread and per grid to maximize performance for a given grid and threads per grid count is something which is not easily determined a priori. In the future, if a specific number of integrations were needed to be performed, this result could be used. The implication of this result, however, is of great importance when considering overall runtime efficiency in a larger global optimization framework. It can be seen that the optimal runtime approached 0.4 seconds as the number of integrations approached 7000. These optimal runtime values were determined using the optimal threads per grid count, but it can be seen that for the least optimal case, the delta of additional increased runtime ontop of the optimal case would more than double the total runtime. So using an incorrect grid sizing could increase runtime for 7000 integrations to nearly 2 seconds. This result motivates the quantification of the optimal

threads per grid value to make sure the GPU and its internal architecture are used appropriately to maximize performance of this application.

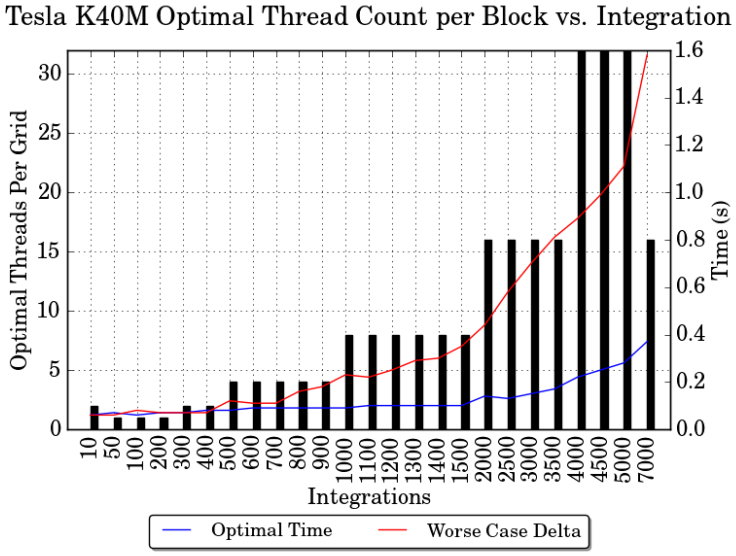

Figure 3.9: Results from detecting the optimal thread count per block for the simple integration problem. It is shown that if the wrong thread count is used, runtime could more than double.

3.4.3 CUDA Streams for Parallel Kernel Execution

To further improve run-time performance, a multiprocessor approach similar to that developed for the global optimization framework was taken during development to allow for distribution of computational workload across a variety of nodes. To accomplish this task, a unique approach to algorithm and program structure was taken which differentiates execution of the code from a serial variant. This paradigm is something which was important to maintain when using CUDA based approaches to improve run-time efficiency. It is not sufficient to simply have a CUDA enabled device, which a given set of processors can access as a computational resource. Rather the code that the CUDA device itself is executing must also be developed to take advantage of the parallelized structure which the entire framework utilizes.

When executing generic CUDA code from a single processor application, the host, being the CPU, sends a command to the device, being the CUDA enabled GPU, to launch a kernel. The flow of this code execution can be

seen in Figure 3.10.

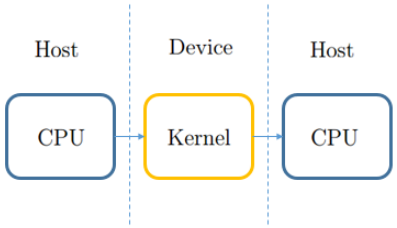

Figure 3.10: An example execution of a single process program executing a CUDA kernel.

If the code that the device is executing finishes faster than an equivalent piece of code that the host variant would run, than a run-time performance increase will be realized if other potential run-time costs are accounted for (e.g. when using a CUDA device, great losses are seen in run-time due to memory allocation and transfer between the host and device). Unfortunately, running this same execution across a variety of processors results in the device effectively locking for each individual kernel call. It only starts processing a new kernel call once the previous one is completed. This behavior can be seen in Figure 3.11.

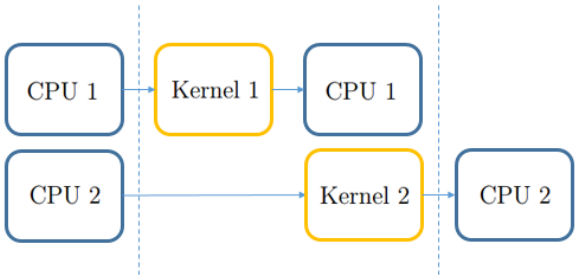

Figure 3.11: An example of a multi-process program executing 2 CUDA kernels, one per process. Notice that the total execution time is delayed due to the linear processing of the CUDA kernels.

This unfortunately produces a significant loss of performance. Luckily, the CUDA architecture has envisioned this problem and a solution has been made available to developers, which has the capability to overcome this problem and execute a number of kernels in parallel. This functionality is provided through a structure called a CUDA stream. Streams are effectively independent serial execution paths which can be populated with different work, where streams basically operate independently of one another as long as the

total resource potential of the device is not reached or shared memory is not required. In this way, each MPI based processor declares its own stream, and is able to realize the flow of execution that would be expected, which can be seen in Figure 3.12.

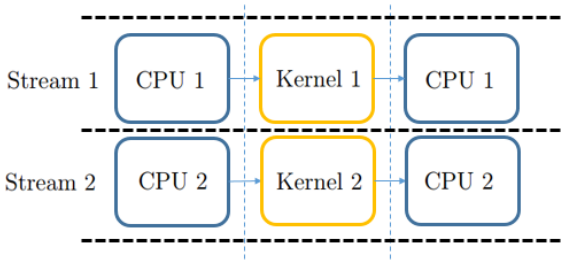

Figure 3.12: An example of a multi-process program executing 2 CUDA kernels on separate streams. In this instance, the CUDA device handles the kernels simultaneously and the program executes in an optimal amount of time.

This is the effective architecture which was developed for integrating CUDA enhanced performance into the operation of the optimization framework. There are a couple pitfalls worth mentioning. First, memory allocation commands are blocking, meaning they force synchronization between the streams. This forces memory allocation before the real parallel execution is performed. Additionally, special asynchronous memory copy functions are to be used, as the standard memory copy functions which are used to move memory between the host and device also force synchronization. Lastly, all kernel and memory transfer calls not assigned to a stream are executed on a default stream. This explains the behavior seen in Figure 3.11, where all the kernel calls were being placed into the default stream where they were being linearly executed.

CHAPTER 4

RESULTS

4.1 Automated Poincaré Preliminary Results

The legend in Figure 4.1 shall be used when visually analyzing results from this section.

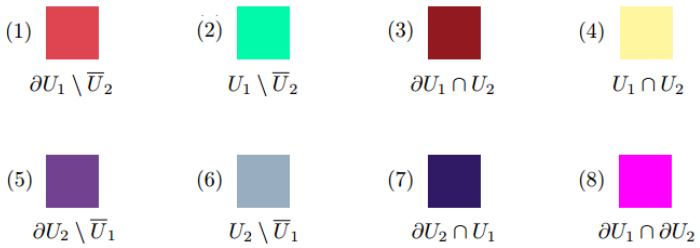

Figure 4.1: The color categorized legend to be used when analyzing algorithm results. Reference Figures 3.3 and 3.4.

Additionally, the following table is referenced in this section.

Table 4.1: Subset Distribution Across Test Cases

Subset	Ideal	EM L_2 to L_1	EM Homoclinic	SE L_2 to EM L_2
1	638	914	1998	1434
2	6295	2456	3767	18023
3	271	388	14	4
4	13503	1752	24	13
5	638	496	2272	124
6	6295	3447	3613	65
7	271	60	15	28
8	6	13	7	4

Table 4.1 shows sizing of grids used in examples to follow.

[1]This chapter contains previously published material from [2] and [5]. The copyright owner provides permission to reprint.

4.1.1 Ideal Case

To begin analysis of how this algorithm performed under a variety of scenarios, a solid baseline and proof-of-concept test was to execute against an ideal dataset. In the case of the spectrum of possible shapes two manifolds could produce when intersecting a Poincaré surface of section, the ideal case was chosen to be the intersection of two circles. The visual result of the algorithm on this ideal dataset can be seen in Figure 4.2.

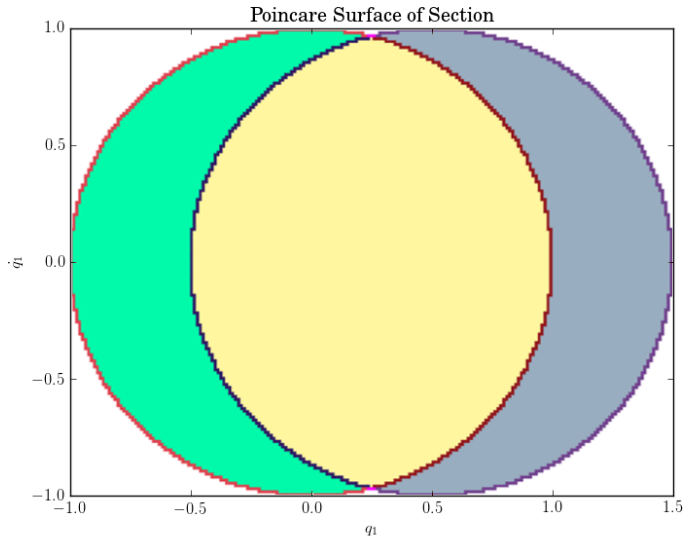

Figure 4.2: The result from an ideal dataset of two circles in the arbitrary $[q_1,\dot{q_1}]$ frame.

Each set defining a circle was made up of 120 points. The algorithm took approximately 87 milliseconds to produce the subsets shown in Figure 4.2. The grid adaptively sized to 221 x 173, providing 38,233 unique grid squares. The result for the distribution of the different regions can be found in table 4.1 in the Appendix section.

4.1.2 Earth-Moon L_2 to L_1 Transfer

Another example which demonstrated the utility of automated detection of Poincaré intersections was to produce an Earth-Moon L_2 to L_1 transfer. Integrating the Earth-Moon primary unstable L_2 manifold $\mathcal{W}_2^{U,i}$ with a Jacobi integral of $C = 3.1720$, and backwards integrating the Earth-Moon L_1 secondary stable manifold $\mathcal{W}_2^{S,e}$ with a Jacobi integral of $C = 3.1881$, there is

a close spatial alignment of the manifold arcs. In this particular case, 100 arcs were propagated for each manifold. The grid was adaptively sized to 198 x 319 for a total of 63,192 grid squares. Overall, the algorithm took 137 milliseconds to define the regions seen in Figure 4.3.

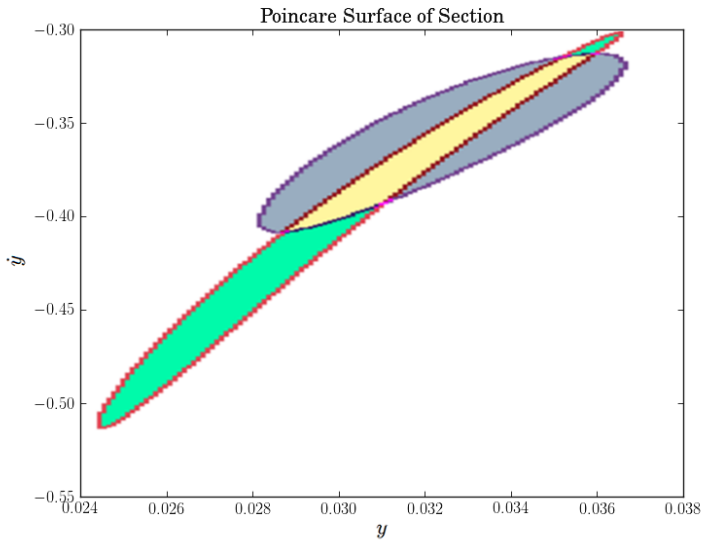

Figure 4.3: The Poincaré section generated from forward integrating $\mathcal{W}_2^{U,i}$ with a Jacobi integral of $C = 3.1720$, and backwards integrating $\mathcal{W}_2^{S,e}$ with a Jacobi integral of $C = 3.1881$.

The distribution of the classification regions can be found in table 4.1. The Poincaré surface chosen was located at the moon $x = 0.9878$ with $y > 0$. In particular, the first intersection with the unstable manifold and the fourth intersection with the stable manifold were used. This produced the result seen in Figure 4.3 when applied to the automated intersection detection algorithm. The bounded regions of the manifolds nicely overlap showing a similar matching of the velocities in at least one axis. From this set, points were randomly generated in the intersection region (8) and resulting states integrated both back onto the L_2 unstable manifold and forward onto the L_1 stable manifold to prove the trajectory matches the mission expectations. A suitable candidate was found with $y = 0.0351$ and $\dot{y} = -0.3148$. Additionally, to close the trajectory, matching of the difference in the Jacobi integral between the two manifolds was accomplished by adjusting the velocity in the x direction. Where the x velocity from the L_2 periodic orbit was calculated to be $\dot{x} = -0.6099$ and to the L_1 period orbit was calculated

as $\dot{x} = -0.5965$. This resulted in a modest Δv of 13.7 m/s, with a single burn being executed at the location of the Poincaré surface. The transfer of this mission was determined to take roughly 33 days. While considerably longer than an impulsive solution to this problem, this transfer embodies the trade-off to be expected when considering a low energy solution. The fully propagated trajectory can be seen in Figure 4.4.

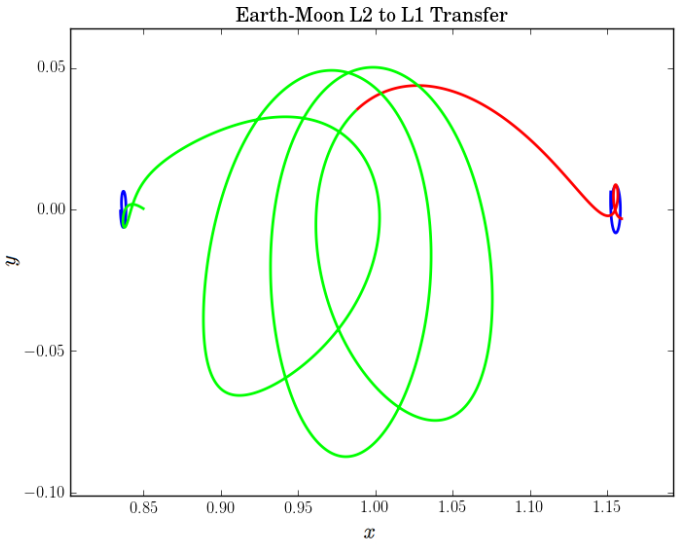

Figure 4.4: Propagation of the L_2 to L_1 transfer both forward and backward from the Poincaré surface. The blue curves represent the L_1 and L_2 periodic orbits with matching Jacobi integrals to the trajectories being propagated. The transfer time is approximately 33 days with a Δv of 13.7 m/s.

Additionally, one may notice in Figure 4.4 that the trajectories match closely to the L_2 and L_1 periodic orbits, but propagate off after some period of time. To match the beginning and final states of the mission to some specific orbit would require additional Δv, but this example characterizes at least the low energy transfer portion of the mission design.

4.1.3 Earth-Moon Homoclinic L_2 Transfer

There exist cases which are challenging for this algorithm to process. One such case is characterized in the following proposed Earth-Moon homoclinic L_2 transfer. In this particular case, the Poincaré surface was again positioned at the moon $x = 0.9878$ with $y > 0$. This is in contrast to a more

standard surface to use for homoclinic transfers at $y = 0$ with $x > 0.9878$. However, the surface chosen produces more interesting Poincaré sections, but this should not be taken as any type of preferred homoclinic transfer. The stable and unstable secondary manifolds, $\mathcal{W}_2^{S,e}$ and $\mathcal{W}_2^{U,e}$ respectively, were integrated with a Jacobi integral of $C = 3.1621$ from the L_2 point. In Figure 4.5, the structure of the Poincaré sections can be seen. Notice the elongated structures which were formed. This produced a challenge at multiple steps in the algorithm pipeline. First, to find any initial internal area was challenging due to the small internal diameter. Additionally, tracing the boundary proved to be challenging if not enough arcs were propagated. If an insufficient amount of intersection points are provided when drawing the boundary region, sections of the boundary region can be cut off and effectively excluded when the nearest neighbor is found. In this case, each manifold had 350 arcs which were integrated to produce a more properly defined boundary region. The grid was adaptively sized to 737 x 836, which produced a grid of 616,132 unique grid squares. While this is an order of magnitude more grid squares than previous examples, this higher level of accuracy was necessary to capture some of the finer detail of the intersection of the two boundaries, which are captured in the popout box inside Figure 4.5. The distribution of grid squares can be found in table 4.1.

Due to the challenges of defining the internal areas, as well as the significantly increased amount of grid squares, the algorithm took 0.898 seconds to complete.

By generating points again inside the intersection region of these two Poincaré sections, trajectories were plotted both forward onto the stable manifold, and backwards onto the unstable manifold in an attempt to close the trajectory back onto the L_2 periodic orbit. One such point generated which achieved this objective was $y = 3.5707$ and $\dot{y} = -0.9569$, where the x velocity was calculated to be $\dot{x} = 3.1918$. The result of this trajectory can be seen in Figure 4.6. One benefit of this transfer is that it takes effectively no Δv to complete, but at the very extreme trade-off of a transfer time of 124 days.

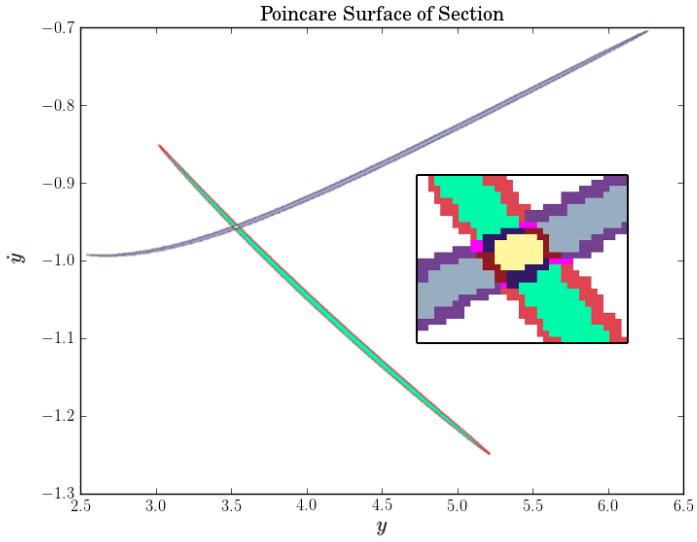

Figure 4.5: An intersection of the L_2 secondary stable and unstable manifolds. The Jacobi integral was calculated to be $C = 3.1621$. Each Poincaré section is comprised of 350 points with the total execution time taking 0.898 seconds.

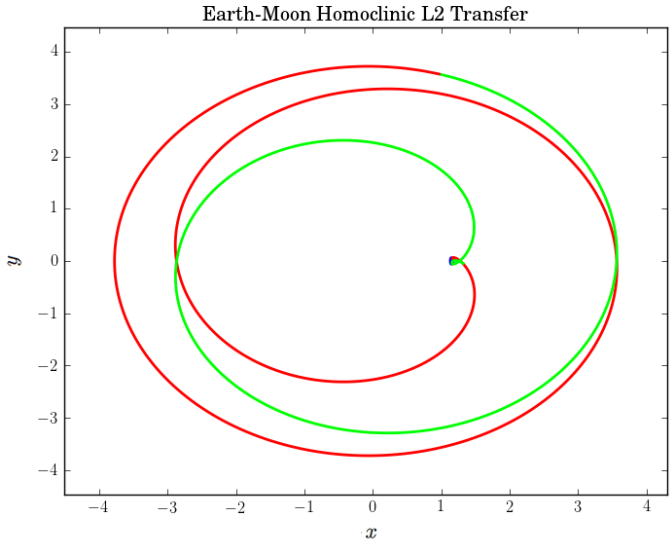

Figure 4.6: The final trajectory produced from a set of $[y,\dot{y}] = [3.5707,-0.9569]$ randomly generated from the intersection region (8) which closes a homoclinic transfer.

4.1.4 Patched 3-Body Sun-Earth L_2 to Earth-Moon L_2 Transfer

To show utility across a variety of scenarios, an application which employed a patched 3-body technique to transfer from the Sun-Earth L_2 to the Earth-

Moon L_2 was developed. In this application, 150 arcs were propagated along the Sun-Earth unstable L_2 manifold $\mathcal{W}_2^{U,i}$ as well as the Earth-Moon stable L_2 manifold $\mathcal{W}_2^{S,e}$. The Sun-Earth manifold was propagated with a Jacobi integral of 3.0007 and the Earth-Moon manifold was propagated with a Jacobi integral of 3.1621. The Poincaré surfaces were generated in the Sun-Earth and Earth-Moon frame separately where the surface was defined at the Earth for $y > 0$. In the Sun-Earth frame, this provided $x = 0.9999$ and in the Earth-Moon frame provided $x = -0.0121$. At this point, the Earth-Moon surface intersection points were converted to the Sun-Earth frame such that they could be compared on an equivalent surface. Processing this combined surface with the algorithm produced the result seen in Figure 4.7.

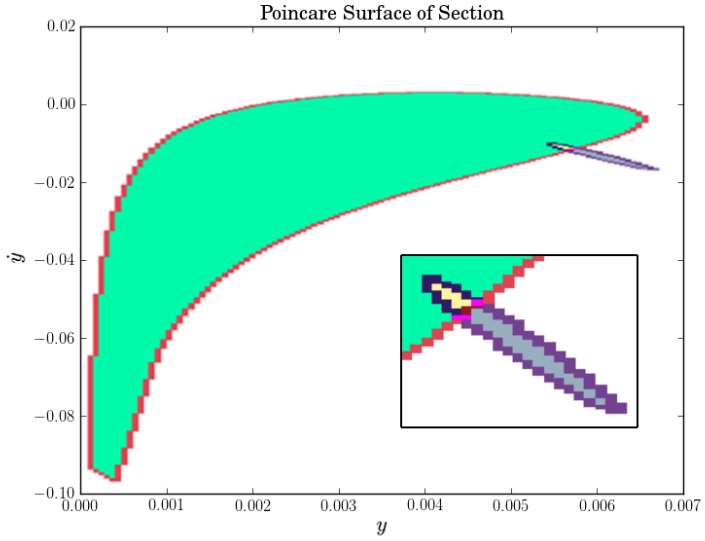

Figure 4.7: A patched 3-body intersection of the Sun-Earth $\mathcal{W}_2^{U,i}$ with the Earth-Moon $\mathcal{W}_2^{S,e}$. These manifolds were propagated with Jacobi integrals of $C = 3.0007$ and $C = 3.1621$ respectively.

The grid adaptively sized to 198 x 728 for a total of 144,144 grid squares. To produce these subsets took approximately 158 milliseconds. This particular case produced a larger grid size than previous cases with a similar number of intersection points due to the varying sizes of the two Poincaré sections. Since the algorithm takes the smallest sizes in both axis directions, the significantly smaller and more compact set causes the larger set to be covered by many more grid squares than necessary. The subsets breakdown can be found in table 4.1. From these regions, the intersection set (8) was queried

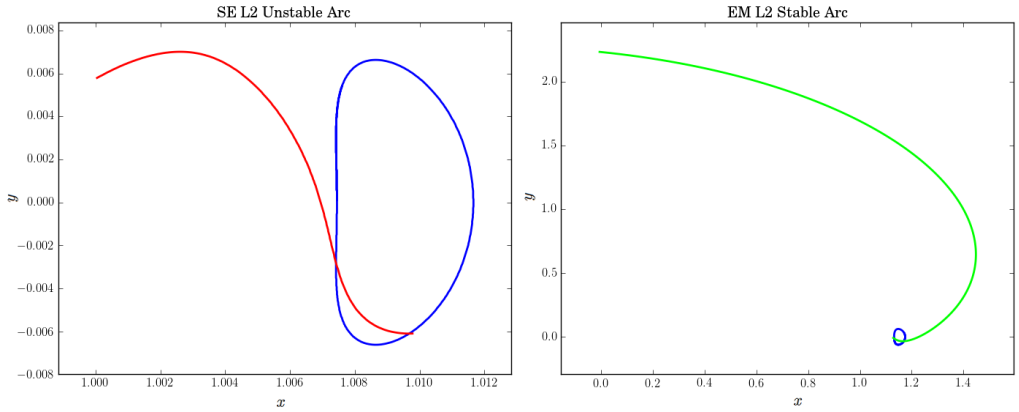

Figure 4.8: Left: Patched 3-body state integrated backward in the Sun-Earth frame onto the Sun-Earth L_2. Right: Patched 3-body state integrated forward in the Earth-Moon frame onto the Earth-Moon L_2.

to produce a $[y,\dot{y}]$ value which closes a trajectory near both the Sun-Earth L_2 and Earth-Moon L_2. A value which meets this criteria was generated where $y = 0.0057$ and $\dot{y} = -0.0112$. To calculate the x velocity for the Sun-Earth arc to fall onto the L_2 manifold using the Jacobi integral provided $\dot{x} = -0.0128$. However, to find the x velocity for the Earth-Moon arc was more involved. The set of $[x,y,\dot{y}]$ in the Sun-Earth frame was transformed into the Earth-Moon frame, and then the x velocity calculated with respect to the Earth-Moon manifolds Jacobi integral in the same frame. In the Earth-Moon frame, this provided $y = 2.2350$, $\dot{y} = -0.3257$, and $\dot{x} = 1.6190$. By integrating these points in the Sun-Earth and Earth-Moon frames independently, the results in Figure 4.8 were generated. The full duration for this transfer was approximately 120 days. Additionally, the maneuver required to accomplish this transfer at the Poincaré surface resulted in a Δv of 71.4 m/s.

4.2 Automated Poincaré Parameter Tuning

While the functionality to autonomously detect and use the intersections of invariant manifolds was produced for this thesis, how to best use two of the major parameters of this algorithm, namely the number of points defining a closed region as well as the number of times the algorithm should attempt to refine the internal area of a bounded region, remains to be explored and

is the purpose of this section.

First an ideal intersection case is explored in an attempt to potentially bound the parameters and better understand the algorithm when it will be applied to more varied cases. The ideal intersection case consists of two intersecting circles, the number of points making up each circle is varied, while also varying the amount of steps the algorithm can take in optimizing internal area of the detected closed region. The demonstration of the results from this exploration can be seen in Figure 4.9 and 4.10.

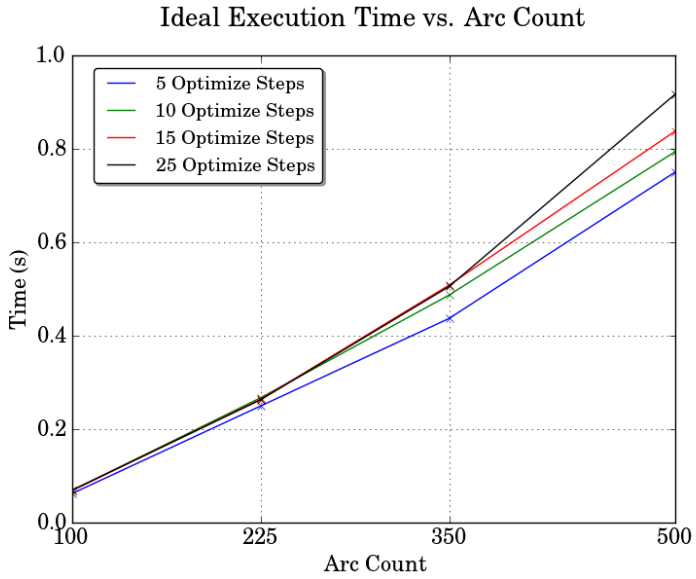

Figure 4.9: Execution time is explored for various counts of points for each set while also varying the amount of times the algorithm can attempt to optimize the internal area of the detected closed region.

It can be seen in Figure 4.9 that, as expected, execution time increases as the number of arcs, or points defining each circle, increases. However, we see that for varying the amount of optimization steps used for refining the internal area, execution time becomes increasingly disjointed as the number of arcs increases. To understand this behavior, Figure 4.10 shows that not until the number of arcs are increased that the opportunity to better refine the internal area can be taken advantage of. With lesser arcs, the bounded regions internal area can only be refined up until some point before gaps in the boundary region are produced. But this comes as a tradeoff, for to produce more refined internal areas, smaller grid sizes must be used, which translates into more grid squares which are needed to be characterized and

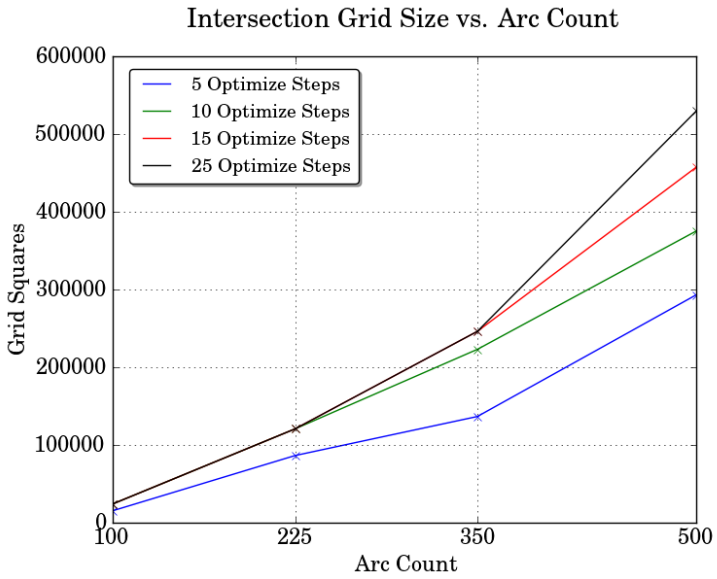

Figure 4.10: Total amount of grid squares, or size, of the intersection grid is presented for various counts of points for each closed region while also varying the amount of times the algorithm can attempt to optimize the internal area of the detected closed region.

thus a longer execution time as well as larger memory usage. Typically for applied problems, an arbitrarily chosen arc count of 350 and optimization step count of 10 have been used. Now that general trends for these variables have been established, how these variables effect results produced by the genetic algorithm and optimization framework at large must be explored. The results from exploring these variables as applied to the optimization framework can be seen in Figure 4.11 and 4.12. For these runs the Hiten inspired Earth to Moon transfer trajectory with manifold to manifold transfer was used. A total of 48 individuals were executed for 200 generations. This process was repeated multiple times and the results averaged in an attempt to minimize bias due to the stochastic searching nature of the optimization framework.

There is an obvious trend in Figure 4.11 that as the number of arcs is increased, the amount of time required to complete a generation also increases linearly. This is due to the integration time required to forward integrate each arc. However, it can be seen that there are diminishing returns as the number of arcs is increased on the total number of manifold to manifold intersections that are detected. With too low of an arc count, some potential bounded

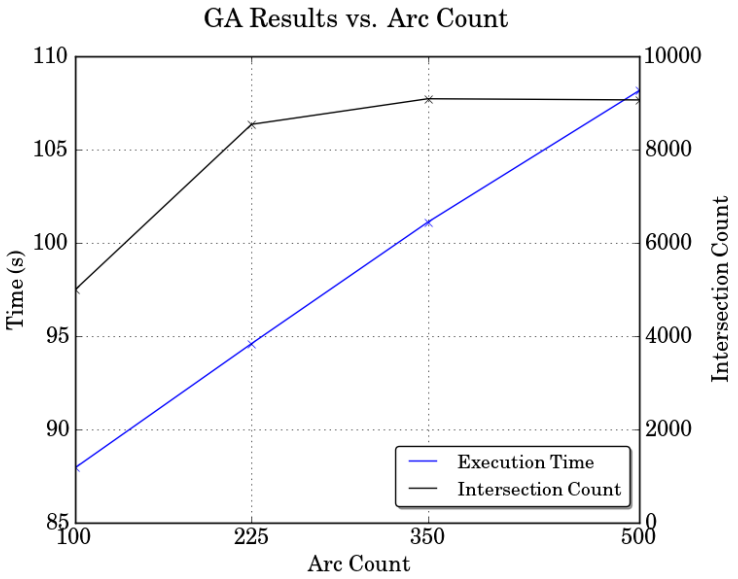

Figure 4.11: This graph presents the average execution time for one generation of the genetic algorithm, as well as the average number of intersections detected from a stochastic run of 200 generations with 48 individuals as the number of manifold arcs defining the boundary region are increased. These runs held the optimize steps variable constant at 10.

regions are missed due to too few points defining the boundary region. As the arc count is increased, the algorithm becomes better at detecting these regions and thus more intersections are detected over the course of the run. From these results, it can be seen that for optimal performance, at least for these specific Sun-Earth to Earth-Moon manifold transfers, that 350 arcs is fully sufficient to explore this space, but 225 arcs could be used with similar results if execution time were of concern. However, this time delta between 350 and 225 arcs can be negated by the application of the CUDA parallelization presented in this thesis. Another important trend can be seen in Figure 4.12. As the number of optimization steps for refining the detected internal area is increased, the algorithm produces more fine meshes, which has already been shown in the ideal intersection case, however, that as the grid becomes more refined, the optimization framework is able to converge more individuals over the total number of generations. This is important from the frameworks perspective as being able to converge individuals more quickly improves the overall effectiveness of the tool, as the genetic algorithm has improved knowledge of the design space. Ultimately, using higher optimize steps is the takeaway from these findings, in addition to the fact that unlike

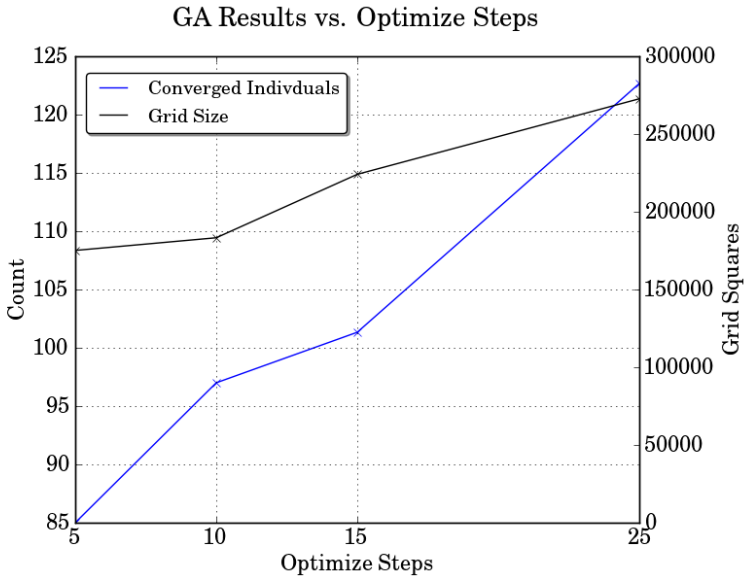

Figure 4.12: This graph presents the average number of converged individuals as well as the average size of the intersection grid derived from a stochastic run of 200 generations with 48 individuals as the number of optimization steps for refining internal area is increased. These runs held the arc count constant at 350.

the increasing the arc count, increasing the optimize steps does not have a measurable impact on the average execution time for a generation.

4.3 CUDA Performance Results

4.3.1 CUDA Integration Performance Improvements

This section is dedicated to exploring the computational time difference between NVIDIA's Tesla K40M and the Intel Xeon X5650 for integration purposes. In Figure 4.13 the runtime across varying numbers of integrations for both serial CPU based and parallelized GPU based approaches is presented. The CPU which was used to produce this plot was the Intel Xeon X5650, which operates at 2.67 GHz, and the GPU used in this computation was the NVIDIA Tesla K40M which operates at 740 MHz.

Linear increase in computational time for both the serial and CUDA cases can be seen. However, we can see that the rate of increase for the serial device is significantly higher. The parallelized hardware on the CUDA device allows

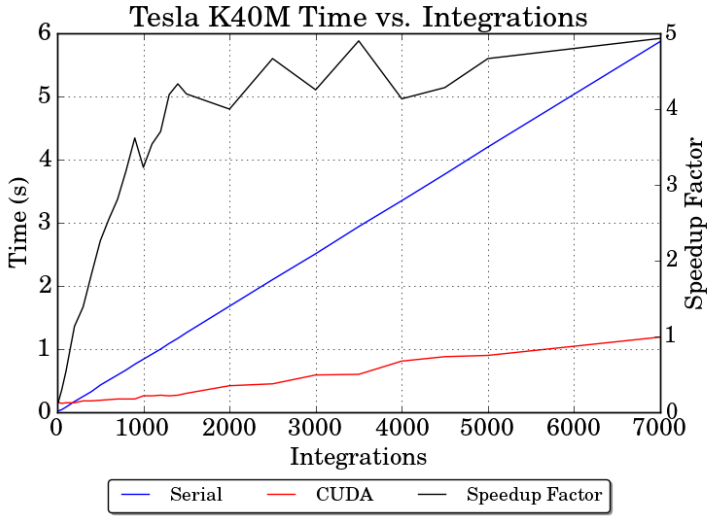

Figure 4.13: Factor of speedup for CUDA vs. Serial integrations. It is shown that improvements top out near 5x speedup.

for significantly more integrations to be calculated simultaneously. Speedup improves significantly between 10 to about 1,000 integrations. At its peak, we see close to 5 times speedup in favor of the CUDA device. However, after about 1,000 integrations the CUDA device saturates and the speedup factor levels out. Warp and thread sizes were considered when launching these kernels, and were tuned on a per integration count basis to maintain maximum performance for each kernel call.

4.3.2 CUDA Manifold Intersection Performance Improvements

The primary application of CUDA for performance increase was the generation of the manifold arcs that are used by the automated manifold intersection portion of the code. This process was discovered to require significantly more run-time than the actual intersection detection itself. By improving performance in this area, the expectation is that solving problems such as the Hiten trajectory, where every evaluation of an individual requires detection of the intersection of two invariant manifolds and hence the propagation of many arcs, could be made more efficient.

To quickly populate the algorithm for automated detection, there are two

areas ripe for parallelization. First, states on the manifold must be generated, where for each arc a state must be integrated for some amount of time on the periodic orbit before a perturbation can be applied. This process was the most time consuming portion which was set out to be parallelized. As an example, while on average the total amount of time to generate one full set of 350 manifold arcs was 2.890 seconds, on average 2.322 seconds of this was required to produce these initial states on the periodic orbit. The second and only other area focused on for CUDA parallelization in this thesis was the following step which integrates these perturbed periodic orbit states until intersection with a Poincaré surface of section. In the timing example used above, 0.555 seconds was used on average to perform this step. This means slightly more than 80% of the computation time was spent generating the perturbed states on the periodic orbit, while 19% of the time was spent integrating to the Poincaré surface of section. To perform an intersection of two invariant manifolds, this process must be reproduced twice, taking an average total time of 5.942 seconds to produce both sets, detecting the intersection, and randomly selecting a state in the region of interest. While these averages demonstrate an overall picture of the timing, Figure 4.14 demonstrates just how much variation can be seen in timing for these different stages, informing why averages were used. While these timings appear minimal for a single manifold intersection case; because the algorithm is being used within a global optimizer, it will therefore be called potentially thousands of times attempting to optimize a single trajectory.

Once the CUDA variants of these portions of the code were produced, the same simulation scenario was performed, with averages again being used to characterize computational timing performance. One major difference in this CUDA variant is that, while the generation of the states on the invariant manifold must still be generated separately requiring two CUDA kernel calls, the integration of the perturbed periodic orbit states onto the Poincaré surface of section has been merged into one kernel call. Because each state is integrated on its own CUDA core, this is the effective change of what was once the performance time cost of 700 integrations in a serial CPU environment to the effective performance time cost of a single integration in a parallel CUDA environment. Even with this consolidation, the results do not show a 700 times speedup. First, there is additional overhead in utilizing CUDA, for example transfer of memory to the CUDA device, both to and from, as

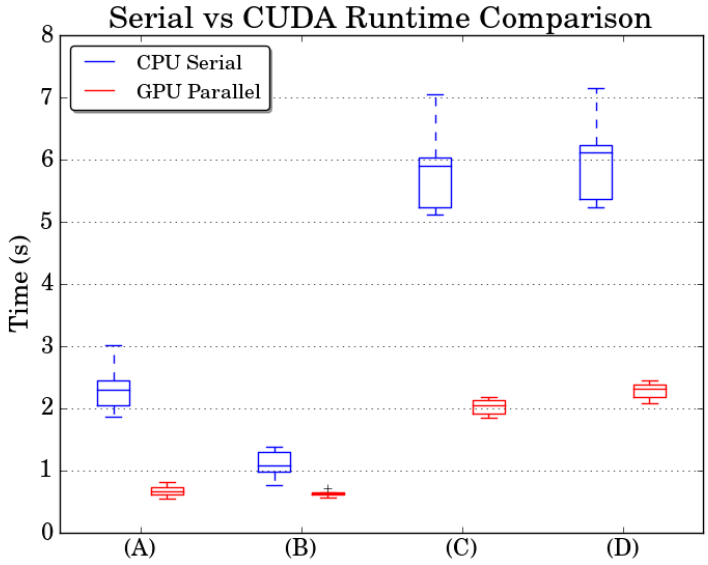

Figure 4.14: Comparison of serial vs CUDA based variants of different portions of the code, characterizing performance gains as achieved through this work. Part (A) shows the time taken to generate the perturbed states on the periodic orbit. Part (B) shows the time taken to integrate the perturbed states onto the Poincaré surface of section. Part (C) shows the total time to finalize both sets of points from two different manifolds. Part (D) shows the total time to return a queried state back to the GA for use by an individual, including detecting intersection of the manifolds.

well as the cost of executing the kernel itself. Another cause for the lesser speed improvement is that CUDA device is running at a slower clock speed. The timing results of this CUDA variant showed an average time to generate the perturbed states on the periodic orbit to be 0.680 seconds, a more than 70% runtime decrease. The integration time for the CUDA variant on average was shown to be 0.629 seconds, which when compared to the total time to integrate both sets of manifold arcs on average, at 1.110 seconds, shows a 43% runtime decrease. Thus the same results which in the serial variant took 5.942 seconds on average was completed by the CUDA variant in an average of 2.294 seconds, which is a decrease in runtime of 61%. Just as in the serial variant, runtime varies depending on the given manifold case to be calculated. A diagram of the various timing values experienced utilizing the CUDA variant can be seen in Figure 4.14. In the author's experience, further performance gains could be realized by reducing the amount of branching

in the CUDA code. Current algorithm implementations for the integrators used have high reliance on conditional statements, which are believed to be hurting performance on the GPU. Overall, these significant performance improvements show that meaningful gains can be realized with CUDA, and that the current effort to use CUDA within the optimization framework is an effective way forward to reduce time to solutions and seeding the automated invariant manifold intersection detection capability.

4.3.3 Hiten-Like Trajectory Results

Since meaningful performance improvements were realized with CUDA, the next step forward was to apply the optimization framework towards solving a problem which requires manifold to manifold transfers, and to determine where future improvements could be realized. To generate a Hiten-like trajectory, the GA provided a set of initial conditions from which to generate potential mission candidates which were then evaluated by SNOPT, as described in Section 3.1. At the beginning of the mission, the spacecraft was specified to start in a circular Earth parking orbit at an altitude of 10,000 km. The spacecraft was then specified to perform a burn to transfer from an Earth parking orbit to connect onto or near a Sun-Earth L_2 unstable interior manifold arc, where the manifold's energy is a parameter chosen by the GA. A transfer onto an Earth-Moon L_2 stable exterior manifold is then executed; again the energy as well as the angle of the Moon at time of intersection being defined by the GA. Finally, a transfer to a circular parking orbit around the Moon at an altitude of 1,000 km was performed. This lunar circular parking orbit is used in substitution for a ballistic capture, which the actual Hiten mission executed. This was selected as the final condition due to easier convergence for the GA, where future iterations could swap in a ballistic capture.

The algorithm finds the manifold to manifold transfer by detecting a manifold intersection between the Sun-Earth and Earth-Moon manifolds using the automated manifold intersection algorithm previously described. Once an intersection is detected, a state in the interior of the Earth-Moon manifold which is external to the Sun-Earth manifold is selected at random, as such, this does not guarantee that the states selected are even useful for the mis-

sion at hand, hence the need for the GA. For further reference, the section queried for this problem is (6) as shown in Figure 3.4. Additionally, with the GA also selecting the energies of the manifolds from which to transfer, as well as the angle of the Moon with respect to the Earth at time of intersection, there is a high probability that the manifolds do not even intersect. Thus it is up to the GA to explore this search space, and effectively discover values for which the Hiten-like trajectory can be closed. It was found on a run with 100 generations and 48 individuals, that about one-third of the candidate individuals trialed actually failed due to the GA defined variables not defining a manifold to manifold intersection. Lowering the count of these failed individuals would improve convergence time of the GA, and will be a major focus going forward. Once a manifold intersection is detected, the randomly generated state is forward integrated in the Earth-Moon frame, and backward integrated in the Sun-Earth frame, taking into account the change in velocity due to the manifold transfer.

To explore this Hiten-like trajectory scenario, the GA was launched with 120 individuals. The final total run-time for 100 generations took 3 hours and 29 minutes. This accounts for the final evaluation and file output which takes place linearly on the head processor. Due to the multitude of solutions which are capable of being explored using this technique, a variety of interesting trajectory candidates were produced. The following examples trade off the two parameters used as objective values in the GA, namely time-of-flight and Δv. The first portion of the trajectory is integrated in the Sun-Earth frame, while the final portion directly after the manifold to manifold transfer is integrated in the Earth-Moon frame, but is transformed back into the Sun-Earth frame for the trajectories presented below.

It should be noted that the results presented in this section are corrected with respect to the calculation of the time of flight and are an amended version of the results shown in [5].

By plotting candidate solutions objective values, one can produce a trade front which characterizes the trade-offs of the two objectives. We can see in Figure 4.18 that as the genetic algorithm advances from generations 50 to 100, the solutions evolve to produce more preferential solutions in both time of flight and Δv. However, it should be noted that this front should not be

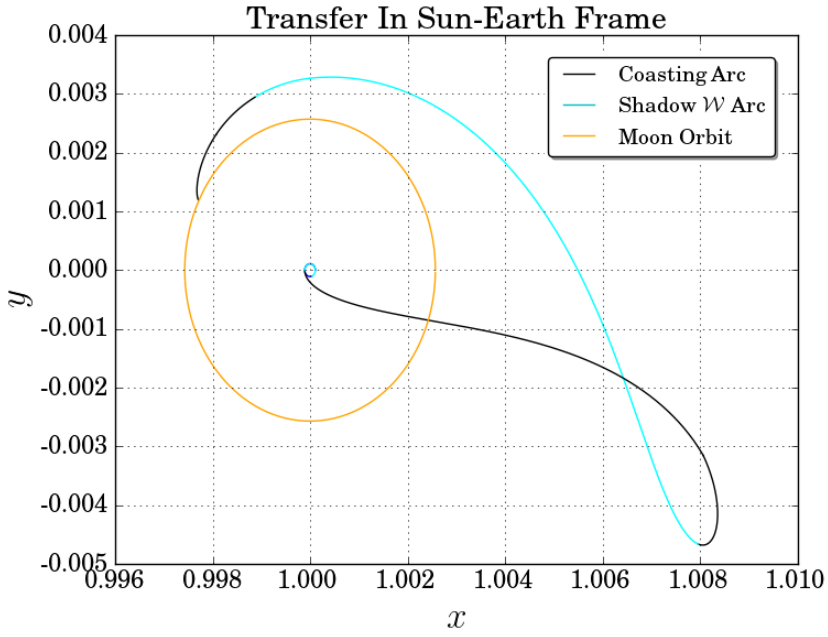

Figure 4.15: One of several Hiten-like trajectory that were solved. The solution shown takes 240 days and has a required Δv of 3.869 km/s to transfer from an Earth parking orbit of 10,000km to a lunar parking orbit of 1,000km. A manifold to manifold transfer was used which required only 60 m/s of Δv.

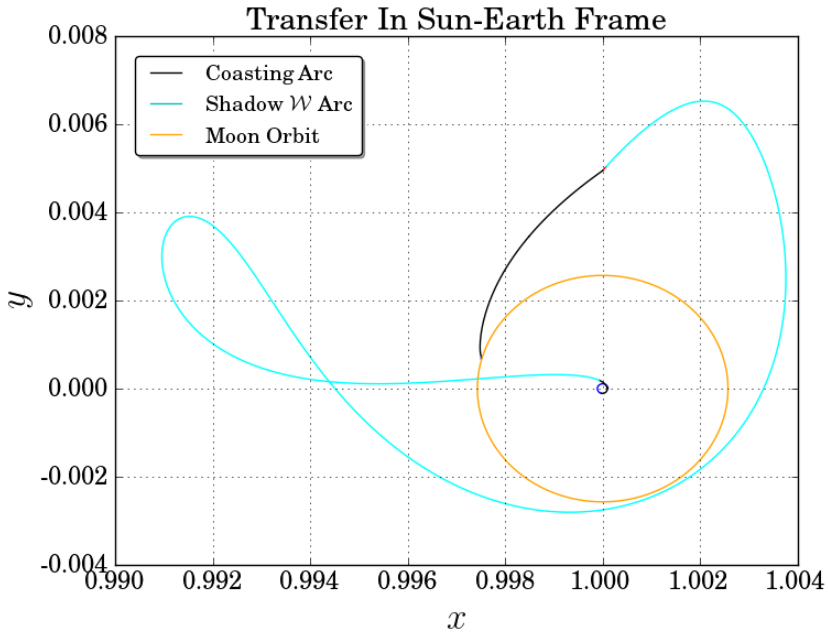

Figure 4.16: One of several Hiten-like trajectory that were solved. The solution shown takes 303 days and has a required Δv of 3.612 km/s to transfer from an Earth parking orbit of 10,000km to a lunar parking orbit of 1,000km. The manifold to manifold transfer in this trajectory required 256 m/s of Δv.

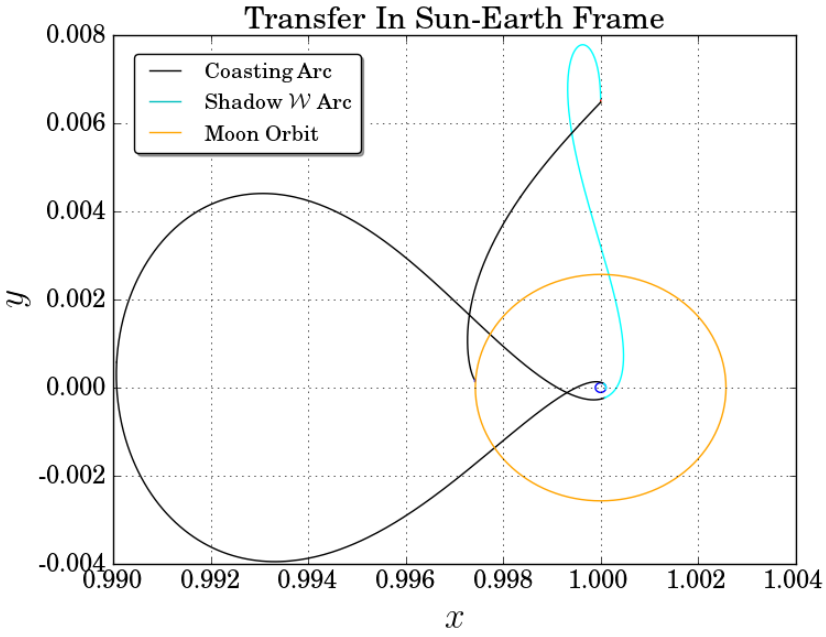

Figure 4.17: One of several Hiten-like trajectory that were solved. The solution shown takes 300 days and has a required Δv of 3.179 km/s to transfer from an Earth parking orbit of 10,000km to a lunar parking orbit of 1,000km. The manifold to manifold transfer in this trajectory required 356 m/s of Δv.

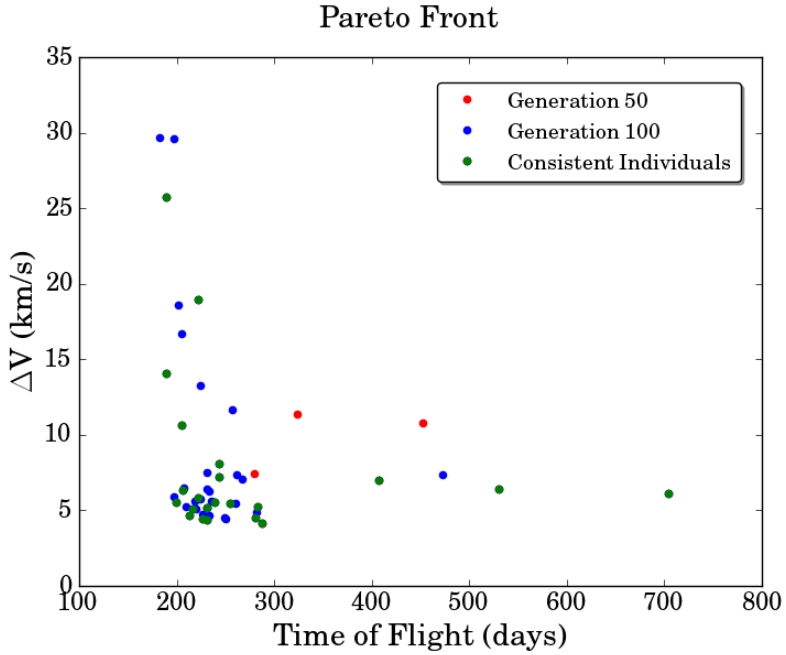

Figure 4.18: An example trade front consisting of 48 individuals. Generations 50 and 100 are shown to demonstrate how the GA evolves better solutions over time.

taken as any final or optimized solutions for this Hiten-like or actual Hiten problem, and are only presented to show the variety of candidates which are capable of being explored by this optimization framework.

Using the resulting candidates as a baseline for nominal Hiten-like trajectory generation performance, the remaining portion of the results section will be used to explore the effectiveness of applying the current CUDA-enhanced Poincaré surface intersection detection to the optimization architecture as a whole.

4.3.4 Global Optimization Performance Improvements

Significant gains in performance were experienced from the CUDA based approach utilized in this thesis. The question of how this translated into improving performance of the optimization architecture at large remains to be answered. To dive into this question, the flow of the GA must be considered, with new bottlenecks being identified. For the GA running in an MPI environment, each individual is expected to be evaluated on its own processor. Thus, the amount of time to produce a generation of N individuals on N processors remains fairly constant, where any performance differences are likely to be seen when performing the competitive stages of the GA, although this will be minimal for small values of N. With this argument being made, we can analyze the timing of a single individual and apply the findings to the GA at large.

On average, the total amount of time for a single generation was found to be 67.965 seconds. This is much larger then the few seconds being considered for CUDA optimization. The majority of the run-time was user defined due to the current implementation of monotonic basin hopping (MBH) in conjunction with SNOPT to perform the local optimization portion of the inner loop. It was desirable as a first analysis to produce many closed solutions which was achieved by providing excess MBH and SNOPT run-times. More efficient local optimization techniques, such as using analytic derivatives and improved algorithm implementations, will allow for better run-time of MBH and SNOPT for these types of problems and still ensure equivalent or improved convergence rates of individuals. Applying the CUDA based approach ended up saving abut 3.6 seconds per individual per generation. Apply this to

the 120 individuals for 100 generations from which these case was executed, and a total of 12 hours of processor time was saved. Another avenue for future run-time improvements would be due to parallel NLP solvers; none are currently available on the commercial market, but some initial results have been shown[13].

With our current method for embedding the automated detection algorithm within the global optimization framework, around one-third of candidate individuals fail to find intersections. This is due to the GA providing poor parameters for manifolds. The Hiten problem implies that giving more control to the automated detection algorithm in this case will promote faster convergence of the GA populations. The timing improvements gained through this approach would then allow for a broader search and improve the probability of returning viable candidate solutions. Another possibility for run-time improvement would be to port the entire automated detection algorithm to a CUDA variant. This could offer interesting capabilities from a optimization framework perspective. For example, in the Hiten problem the GA is supplying an energy for each of the potential manifold candidates, as well as the angle for the Moon in the case of the Earth-Moon manifolds intersection with the Poincaré surfaces of section. But if the GA instead only supplied the two energies, and entire sets of potential candidates across angles were simultaneously calculated on the CUDA device, then only potential results which actually have an intersection would be used. This would effectively remove the possibility of having a wasted individual evaluation based upon the angle variable. Depending on the computational and memory requirements on the CUDA device, the same concept could be further applied to one of the energy variables for calculating simultaneous Poincaré intersection detections. These concepts will be the focus for future performance improvements.

CHAPTER 5

CONCLUSION

This thesis explored the intersection of invariant manifolds as applied to low energy trajectories from the theoretical perspective, algorithmic implementation for their detection and classification, as well as application and utility to trajectories in the three-body problem. The background for the presented algorithm and its application were reinforced by the exploration of the generation of invariant manifolds, as well as the Hiten mission and its motivation on this work. An explanation of the optimization framework which was utilized to produce trajectory results was presented, and a thorough explanation of the automated Poincaré intersection capability and algorithm were explored. Additionally, the need for and pursuit of speed improvements through CUDA based parallelization were discussed and results presented which shows that CUDA parallelization is a viable candidate for quickly producing states to be used to populate Poincaré surfaces of section. Proof of concept results and tuning of the internal area refinement as well as the number of manifold arcs utilized for the presented algorithm were discussed and explored. Next step improvements for this work were also presented in the results, where it is suggested that for the generation of patched three-body trajectories between the Sun-Earth and Earth-Moon frames, that the genetic algorithm be given less control over the angle of the Moon and energies of the invariant manifolds. In this way, CUDA parallelization for a broad search implementation should be used to better utilize CPU resources and minimize individuals which cannot be closed due to bad parameter selection.

CHAPTER 6

FUTURE WORK

Two main avenues are presented in this section, which are the next step with respect to future work for this thesis. The first topic to discuss is general speed improvements for the algorithm. Currently, an equally spaced grid structure is used to bound the set of the manifold intersections. However, it can be seen in the preliminary results explored using the algorithm presented in this thesis that when two manifold intersections of varying grid sizes are compared, the utilization of this data structure can yield an undesirably large set of grid squares, and hence memory usage and processing requirements for producing results. After implementation of this algorithm and looking back on its output, there seems to be a better data structure to capture this space, since there exist areas of high concentration of points, but also areas with large empty spaces which do not need to be explored in detail. The data structure which comes to mind is the quadtree. This data structure has use for a similar purpose in the application of computer graphics, specifically for terrain rendering optimization. For a given area, the quadtree subdivides around a point, or for the case of this algorithm a set of points, turning the starting area into four equivalent smaller areas. This process is repeated to some depth. This produces the property that accuracy around points of interest are of high fidelity, and areas which are lacking points, which for the cases of this algorithm are perhaps the interior or exterior regions of a manifold intersection, are rendered in significantly less detail. This would produce the net result of less elements to define a manifold intersection, meaning faster processing times and more efficient memory usage.

Given that the maximum grid size of the algorithm presented in this thesis was set to be 4096 x 4096, or 2^{12} x 2^{12}, this presented a maximum possible number of grid squares of 16,777,216. Using the quadtree data structure, a depth of 12 would be needed to ensure the same accuracy as the 4096 x 4096

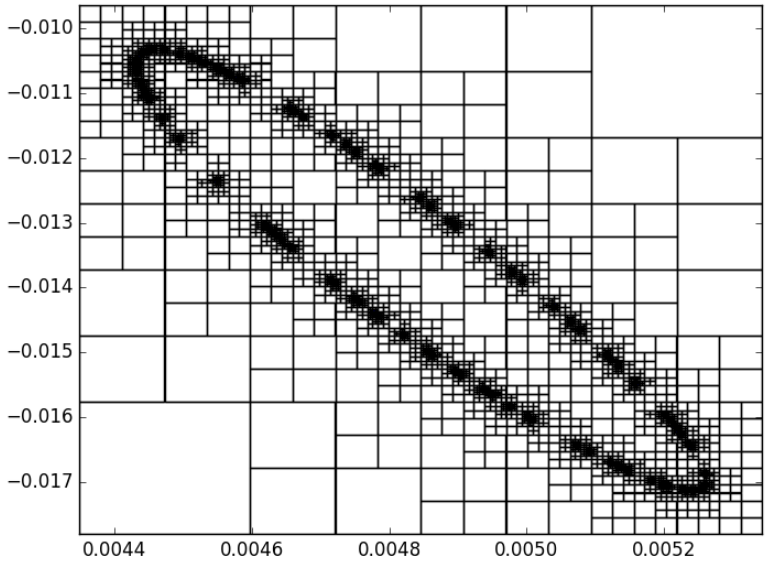

Figure 6.1: An example quadtree implementation with 350 points generated from an Earth-Moon L_2 stable exterior manifold which has been transformed into the Sun-Earth frame. The depth of this quadtree was set to 12.

grid around the points of interest. In Figure 6.1 an example test of a quadtree is presented to show the massive amount of memory savings which can be seen using this data structure. To generate this, a depth of 12 was used. Additionally, a set of points similar to those found in this thesis were supplied. The total runtime to produce this set, without classifications of internal or external regions, took 68 milliseconds. What is particularly impressive however, was that instead of more than 16 million elements to define the grid, only 7861 terminal nodes in the quadtree were needed to produce this result. While this method requires further testing and development in the future, this simple example demonstrates the massive savings which could be realized using this data structure to define regions of intersection for invariant manifolds.

Outside of speed improvements for this algorithm, there is room for applications to more complex problems than just the CR3BP. Inside a more complicated 4-body model, or higher fidelity solution, the invariant manifolds which are so heavily relied upon in this work do not exist. To expand

into this regime, the use of dynamical structures called Lagrangian Coherence Structures (LCS) would be useful to be explored. The current implementation of this algorithm would not be able to tackle exploring the dense clouds of points which are generated with producing a LCS. But, although the problem is significantly harder to automate, an algorithm which could identify regions of interest within the LCS map would be extremely useful as a mission design tool.

REFERENCES

[1] V. Shah, R. Beeson, and V. Coverstone, "A method for optimizing low energy transfers in the earth-moon system using global transport and genetic algorithms," *AIAA/AAS Astrodynamics Specialist Conference, AIAA SPACE Forum, (AIAA 2016-5438)*, 2016.

[2] J. Aurich, R. Beeson, and V. Coverstone, "Automated detection of invariant manifold intersections using adaptive grid method," *AIAA/AAS Astrodynamics Specialist Conference, AIAA SPACE Forum, (AIAA 2016-5437)*, 2016.

[3] R. Beeson, D. Bunce, and V. Coverstone, "Approximation methods for quick evaluation of invariant manifolds during global optimization," *AIAA/AAS Astrodynamics Specialist Conference, AIAA SPACE Forum, (AIAA 2016-5439)*, 2016.

[4] V. Shah, R. Beeson, and V. Coverstone, "Automated Global Optimization of Multi-Phase Trajectories in the Three-Body Problem Using A Varaible Gene Transcription," in *AAS/AIAA Space Flight Mechanics Conference*, San Antonio, TX, Feb. 2017.

[5] J. Aurich, R. Beeson, and V. Coverstone, "CUDA-Enhanced Integration for Quick Poincare Surface Intersections in a Global Optimization Framework for Low Energy Transfers," in *AAS/AIAA Space Flight Mechanics Conference*, San Antonio, TX, Feb. 2017.

[6] D. Bunce, R. Beeson, and V. Coverstone, "Resonance Orbit Generation for Global Optimization Seeding," in *AAS/AIAA Space Flight Mechanics Conference*, San Antonio, TX, Feb. 2017.

[7] M. Dellnitz, O. Junge, M. Post, and B. Thiere, "On target for venus set oriented computation of energy efficient low thrust trajectories," *Celestial Mechanics and Dynamical Astronomy Celestial Mech Dyn Astr*, vol. 95, pp. 357–370, 2006.

[8] A. Zanzottera, G. Mingotti, R. Castelli, and M. Dellnitz, "Intersecting invariant manifolds in spatial restricted three-body problems: Design and optimization of earth-to-halo transfers in the sunearthmoon scenario," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, pp. 832–843, 2012.

[9] W. S. Koon, M. W. Lo, J. E. Marsden, and S. D. Ross, "Dynamical systems, the three-body problem and space mission design," 2006.

[10] J. S. Parker, "Low-energy ballistic lunar transfers," M.S. thesis, University of Colorado, 2003.

[11] J. Masdmont and E. Canalias, "Homoclinic and heteroclinic transfer trajectories between planar lyapunov orbits in the sun-earth and earth-moon systems," *Discrete and Continuous Dynamical Systems DCDS-A*, vol. 14, pp. 261–279, 2005.

[12] W. S. Koon, M. W. Lo, J. E. Marsden, and S. D. Ross, "Low energy transfer to the moon," *Celestial Mechanics and Dynamical Astronomy, Vol. 81*, 2001.

[13] A. Ghosh, R. Beeson, D. Ellison, L. Richardson, D. Carroll, and V. Coverstone, "A non-linear parallel optimization tool - nlparopt," in *AAS/AIAA Astrodynamics Specialist Conference, Vail, CO*, no. 15-808, August 2015.