A LEARNING APPROACH TO SAMPLING OPTIMIZATION: APPLICATIONS IN ASTRODYNAMICS

A Dissertation

by

TROY ALLEN HENDERSON

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2013

Major Subject: Aerospace Engineering

ABSTRACT

A new, novel numerical optimization algorithm is developed, tested, and used to solve difficult numerical problems from the field of astrodynamics. First, a brief review of optimization theory is presented and common numerical optimization techniques are discussed. Then, the new method, called the Learning Approach to Sampling Optimization (LA) is presented. Simple, illustrative examples are given to further emphasize the simplicity and accuracy of the LA method. Benchmark functions in lower dimensions are studied and the LA is compared, in terms of performance, to widely used methods.

Three classes of problems from astrodynamics are then solved. First, the N-impulse orbit transfer and rendezvous problems are solved by using the LA optimization technique along with derived bounds that make the problem computationally feasible. This marriage between analytical and numerical methods allows an answer to be found for an order of magnitude greater number of impulses than are currently published. Next, the N-impulse work is applied to design periodic close encounters (PCE) in space. The encounters are defined as an *open rendezvous*, meaning that two spacecraft must be at the same position at the same time, but their velocities are not necessarily equal. The PCE work is extended to include N-impulses and other constraints, and new examples are given. Finally, a trajectory optimization problem is solved using the LA algorithm and comparing performance with other methods based on two models-with varying complexity-of the Cassini-Huygens mission to Saturn. The results show that the LA consistently outperforms commonly used numerical optimization algorithms.

DEDICATION

To Melissa, Joshua, Caleb, Simon, and Benjamin...my inspiration and my joy.

ACKNOWLEDGEMENTS

Completing any long journey in life puts one through a wide range of trials and emotions. Along such a journey, one is blessed to have friends and advisors who help guide them along their path. Many people have been a part of my life, shared in this journey, and sustained my effort in different ways. To say a simple "thank you" to any of them is severely deficient.

Dr. Daniele Mortari has served as advisor, mentor, colleague, and (perhaps most importantly) friend. I cannot begin to thank him enough for his many invaluable contributions in any of these roles. Thank you for sharing ideas, listening to my ideas (even the bad ones), your inspiration, and your enthusiasm for life. I look forward to working with you for many more years to come.

To have worked with and studied under Dr. John Junkins is itself a blessing. Thank you for sharing your love of astrodynamics and letting me pick your brain. I know that you find satisfaction in seeing your students succeed. Thank you for your advice and foresight.

Dr. James Turner has been an invaluable sounding board and a nearly infinite well of professional advice. Dr. Maurice Rojas graciously agreed to sit on the committee and has provided significant resources to the mathematical developments in this dissertation. Dr. Martín Avendaño, although not on the committee, has made much of this work possible through lending his mathematical insight and friendship.

This journey would have been rough without colleagues and friends like Dr. Drew Woodbury, Dr. Jeremy Davis, Chip Hill, Dr. Thomas Talley, Pedro Davalos, Dr. Thomas Pollock, Dr. Christian Bruccoleri, Dr. Anup Katake, and Dr. Gianmarco Radice who made work fun.

I would be remiss to ignore those who made a contribution to my life outside of work: John and Amanda Fellers, Tony and Melissa Love, Bob Morris, and the body of Parkway Baptist Church.

Finally, I wish to acknowledge my family, Melissa, Joshua, Caleb, Simon, and Benjamin for their extreme patience, unending forgiveness, and unconditional love. On to a new beginning...

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

Many important problems are posed as a mathematical optimization problem. Optimization is the process of determining the value (or values), from a set of design variables and constraints, that provide a solution minimizing a cost function which defines optimality of the given problem.

Numerical optimization is a required tool for solving complex problems in astrodynamics,[1] including: spacecraft design, mission design, trajectory optimization, orbit determination, and spacecraft maneuvers, to name a few. Continuous problems are readily handled, however, traditional, calculus-based optimization techniques begin to fail or become intractable when applied to some problems in astrodynamics, especially those with numerous local extrema or discontinuities.

Many optimization algorithms exist and are based on different mathematical concepts. There is no optimization technique that works best across all problems or all dimensions. Current optimization techniques are classified as follows:

- **Analytic**: Gauss-Newton, gradient based
- **Deterministic**: branch methods, algebraic geometry
- **Stochastic**: Simulated Annealing, Monte-Carlo
- **Heuristics**: Genetic algorithm, Particle Swarm Optimization

Analytic methods extremize a function by setting the gradient of the function equal to zero. The resulting set of nonlinear equations is solved and provides the extremal locations. These methods have been broadly studied are have their strongest appeal

[1] *Astrodynamics* is defined as the application of celestial mechanics applied to the problem of spacecraft motion.

in the solid mathematical foundations upon which they have been developed. The stochastic and heuristic methods are more described in Chapter II.

The problems arising in astrodynamics are, in general, highly nonlinear, potentially discontinuous in time and space variables, and may required both integer and real values. Furthermore, there are frequently more than one local extremum, so the issue of local versus global solutions arise often. Thus, classical analytical approaches are often ill-suited for solving these problems. Given the discontinuous nature of these problems, it remains unlikely that deterministic algorithms for the global optimization of general functions can be derived[1]. Non gradient-based numerical optimization approaches have recently seen success in solving complicated engineering problems. The method developed in this work seeks to bridge the capability gap between popular heuristic algorithms and applications in astrodynamics.

The motivation for seeking a new method came when Henderson and Mortari were solving an optimization problem in which the cost function to be minimized had the shape shown in Fig. 1.1. Figure 1.2 shows a two-dimensional cross-section, projected to the $x - z$ plane. As can be seen, the minima lies at about $x = 3,100$, however, there exist macro- and micro-scale structure to this function that makes it very difficult to numerically optimize. In addition, the function is discontinuous so gradient-based methods were not attempted and many popular heuristic methods quickly fell into a local minima.

This dissertation is aimed at introducing an unconventional non gradient-based sampling method for numerical optimization method by developing the theory, demonstrating the algorithm, and testing the method on a set of problems in astrodynamics. Here, a new numerical optimization technique called the Learning Approach to Sampling Optimization (LA) is presented. The LA algorithm uses *rejection sampling* to recursively learn the distribution from which samples are drawn, forcing the point

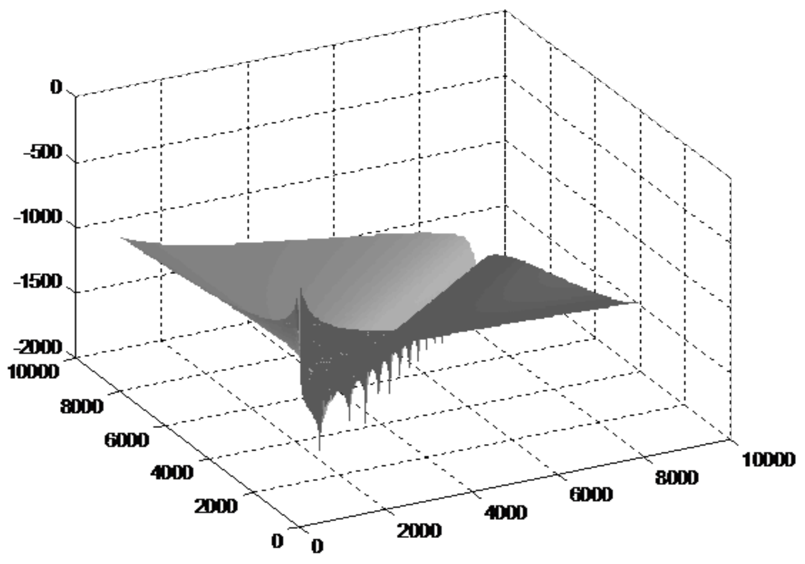

Figure 1.1: 3-D Cost Function

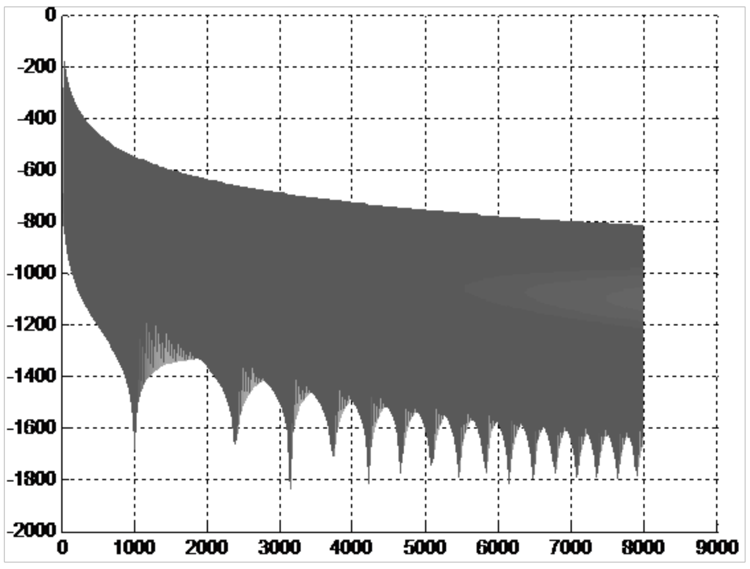

Figure 1.2: 2-D Cross Section

selection toward the sought extrema. The theoretical development of the LA and comparison with other algorithms on a set of benchmark functions is made. Three significant applications for the aerospace community are then provided. The development of the problem and solution is given and performance comparisons are made with more standard algorithms. The main objectives of the research leading to this dissertation are:

- To develop a numerical optimization method that is based on a sound mathematical foundation;
- To compare the performance of the algorithm with currently used algorithms;
- To pose new problems in the field of astrodynamics;
- To solve the newly posed problems and compare the solutions with other algorithms where possible.

These goals are achieved by studying three problems from the field of astrodynamics. The first problem investigates the N-impulse orbit transfer and rendezvous problem. Here, analytical bounds are derived that constrain the choices of velocities, which allows an optimization algorithm to find a reasonable solution for a high number of variables. Examples are given that demonstrate a significant improvement over currently published results.

The second problem investigated is periodic close encounters in space. This problem uses the results of the N-impulse transfer in a new application. Periodic close encounters occur when one spacecraft periodically encounters the orbit of a second spacecraft under specified conditions. The purpose of this problem is for observation and mission or rendezvous planning.

Finally, the new optimization technique is applied to a benchmark spacecraft trajectory design problem that has been posed by the European Space Agency (ESA) and has several published solutions. The trajectory studied is similar to that of the actual trajectory of the Cassini-Huygens spacecraft route to Saturn. Statistics are given and compared with known solutions.

The dissertation documents the results of the research organized as follows:

Chapter II briefly introduces the basic concepts of optimization and astrodynamics required to understand the theory and applications by a non-specialized reader. This chapter provides the basic mathematical definitions, a brief review of several popular numerical optimization algorithms, and a basic introduction to definitions and equations from astrodynamics. References are given for further study.

In Chapter III, the theoretical development of the Learning Approach method is given. Once the algorithm is shown, a proof of convergence is given for a specific set of functions and numerical tests confirm the correctness. A series of benchmark functions are tested and performance comparisons are made with popular algorithms. Finally, consideration is given to the implementation of the algorithm.

Chapter IV introduces three applications taken from astrodynamics to be solved. For each problem, the significance of the problem is given, previous work is discussed, each problem is defined and then solved. Performance comparisons are again made with popular algorithms. In addition, new results are given where applicable.

Finally, in Chapter V, the results are summarized and discussed to highlight the positive aspects and limitations of the new LA algorithm, laying groundwork for continuation and improvement of the present work.

The main contributions made to the astrodynamics and numerical optimization communities by this dissertation are a new algorithm, the LA, which is demonstrated to perform faster and more accurately than several popular heuristic methods, and new problems are framed and solved in astrodynamics, and previously studied problems are pushed to N-dimensions. The LA is used to solve a major benchmark trajectory optimization design problem to within a few percent of the best known algorithm using a laptop, whereas the best solutions were found by research groups with access to small supercomputers and software specifically designed for interplanetary trajectory optimization.

2. REVIEW OF NUMERICAL OPTIMIZATION AND ASTRODYNAMICS

This chapter introduces a variety of numerical optimization-related mathematical definitions and algorithms along with some fundamentals of astrodynamics which are required for a complete understanding of the remainder of the dissertation. Terminology and notation will be introduced for the subjects of numerical optimization and astrodynamics. This chapter provides an introduction to broad areas of numerical optimization and astrodynamics, specifically involving stochastic algorithms. References for further study are provided as each concept is introduced.

2.1 Introduction to Optimization Theory

Perhaps the most concise and broadly applicable definition of optimization is given by Rao[2] : "Optimization is the act of obtaining the best result under given circumstances." In other words, optimization is the process by which a design point is found that satisfies given constraint criteria and extremizes (i.e., minimizes or maximizes) some measure important to the engineer. The mathematical theory of optimization can be traced back to giants such as Newton, who developed calculus, Gauss, who developed the steepest descent method, and Lagrange, who introduced the Lagrange multipliers in early constrained optimization problems.

An optimization problem is formally defined (following the notation in Reference [3]) by a compact and countable search space, $\Omega \subset \Re^N$ (with $0 < N \in \mathbb{Z}$), and a cost function (also commonly called a fitness, loss, or objective function), $J(\mathbf{x}) : \Omega \to \Re$. Global optimization methods search for the global optimum in a feasible search space Ω. Many methods encounter difficulty when the optimal value is on an edge of the search space. This shows how critical the definition of Ω is for solving a problem.

For this work, there are no requirements on the smoothness, convexity (although assuming it to be *nonconvex* is more general), or continuity of the function $J(\mathbf{x})$. In general, real-world problems, $J(\mathbf{x})$ may not be given in a closed form, meaning $J(\mathbf{x})$ may be based on a simulation model[4].

The search space, Ω is defined by a set of N-dimensional points, $\mathbf{x}$, of which one (or more) point(s), $\mathbf{x}^* \in \Omega$, exists such that the cost function is optimized. Here, as in all practical cases, the search space, Ω, is bounded even if, in theory, Ω does not need to be bounded. For example, the search space may be bounded (i.e., discretized) by the number of digits used in a given computer software (e.g., double precision floating-point). Constraints on the optimization problem may be present for a given problem in the form of inequality and equality constraints[5] such that

$$\begin{cases} c_j(\mathbf{x}) \leq 0 & j \in I \\ c_j(\mathbf{x}) = 0 & j \in E \end{cases} \tag{2.1}$$

where the functions $c_j(\mathbf{x}) : \Re^n \rightarrow \Re$ for $j = 1, ..., m_I + m_E$, the number of constraints is $m_I + m_E$, and I and E are a disjoint set of integers of cardinalities m_I and m_E respectively. Usually, in real-world problems, $J(\mathbf{x})$ with constraints is not analytically treatable[4]. However, for the present discussion, only unconstrained problems are considered (the ranges of the variables will be considered bounded, defining Ω). Constraints will be added using a penalty method which will be detailed in the description of the problem.

The search space Ω may be any arbitrary metric space, whereas for engineering problems, generally only $\Omega \subset \Re^N$ is considered. Allowing Ω to be specified as any metric space allows for more sophisticated data structures to be used in the optimization process. For example, if real, integer, and Boolean design parameters could all be used in the same problem, it would be very convenient to define the data structure, $\mathbf{x}$, to contain the appropriate values than to transform all parameters to real values. For this reason, this dissertation will use real-encoding of variables as opposed to binary encoding as the problems discussed are real-valued.

Remark 2.1.1. *Another class of problems is an* optimal control *problem where the variable,* $\mathbf{x}$*, becomes a time-varying trajectory,* $\mathbf{x}(t)$*, for* $t \in [0, T]$*. The optimal control problem is similar to the present formalism after the discretization of* $[0, T]$*.*

The purpose of the optimization algorithm is to automatically find for the user the global optimum value(s). Minimization will be considered for the following descriptions, derivations, and application problems. It should be noted, however, that this does not restrict the generality of the problems shown, or of the algorithms described since the following obvious identity holds[6]

$$\max\{\mathrm{J}(\mathbf{x})|\mathbf{x} \in \Omega\} = -\min\{-\mathrm{J}(\mathbf{x})|\mathbf{x} \in \Omega\} \tag{2.2}$$

Definition 1. *A point is a* local minimum *if the value of the cost function,* $J : \Omega \to \Re$*, is lower than the cost function value of the points in a small, feasible neighborhood around it.*

Mathematically, a point $\mathbf{x}^*$ is a local minimum if

$$J(\mathbf{x}^*) \leq J(\mathbf{x}) \qquad \forall \mathbf{x} \in N_\epsilon(\mathbf{x}^*) \tag{2.3}$$

where

$$N_\epsilon(\mathbf{x}^*) = \{\mathbf{x} \in \Omega, \|\mathbf{x} - \mathbf{x}^*\| < \epsilon, \epsilon > 0\} \tag{2.4}$$

defines the ϵ-neighborhood of $\mathbf{x}^*$ and the norm in Eq. (2.4) is taken to be the

Euclidean norm. It should also be noted that the requirement exists that

$$J(\mathbf{x}^*) > -\infty$$

For a maximization problem, simply change the '$\leq$' sign to '$\geq$' in Eq. (2.3) and similarly the following must be true

$$J(\mathbf{x}^*) < \infty$$

Definition 2. *The* global minimum *is the value of* $\mathbf{x}^*$ *that gives the minimum value of the cost function over the entire* N*-dimensional search domain,* Ω*.*

Mathematically, a point x^* is the global minimum if

$$J(\mathbf{x}^*) = \min_{\mathbf{x}\in\Omega} J(\mathbf{x}) \tag{2.5}$$

In general, numerical optimization methods can be categorized in two ways: deterministic and stochastic[7]. Purely deterministic algorithms seek to define a neighborhood in which it is guaranteed that the global minimum is located, and in some cases find the global minimum by an exhaustive search over Ω. In order to guarantee the success of a deterministic method, further assumptions on the cost function are generally required. Deterministic methods are generally gradient based and require the assumption of continuity and differentiability of the cost function (although not all methods require these conditions). One popular approach is to assume the *Lipschitz continuity condition* provides an upper bound on the rate of change of the function over all of the search space, Ω. A constant, L, is given such that for all

$\mathbf{x}, \tilde{\mathbf{x}} \in \Omega$ such that[8]

$$|J(\mathbf{x}) - J(\tilde{\mathbf{x}})| \leq L \, ||\mathbf{x} - \tilde{\mathbf{x}}|| \tag{2.6}$$

This is difficult to guarantee in practice. Deterministic methods are more suitable for local optimization and require a suitable initial condition within the region of convergence. The traditional mathematical treatment of global optimization problems requires restrictive assumptions be made on the problem in order to obtain a solution. Derivative based minimization obviously implies continuity and an associated basin of attraction of a local minimum. When these methods converge, the frequently are efficient in accurately localing local minima. However, their utility for multimodal and/or discontinuous problems is limited. Thus, the traditional, deterministic solution methods are not very reliable for general global optimization problems. These requirements and conditions have motivated the recent derivation and widespread use of highly effective, stochastic methods.

No method or software currently exists to efficiently solve *all* global optimization problems. Thus, it is natural to involve stochastic and heuristic parameters and methods in the algorithms. Stochastic methods have the downside that only a probabilistic (asymptotic) guarantee[8] is made that the global minimum value is found; however, a common practice is to start a local optimization routine starting from the best point(s) found by the global optimization. These optimization methods incorporate random elements, generally in the point selection process. Törn says "The algorithms of multidimensional global minimization are rather complicated. Their realization includes heuristic procedures for solving various auxiliary subproblems."[9] The efficiency of many stochastic algorithms is strongly dependent on the heuristically chosen tuning parameters governing a problem formulation[9].

Unconstrained stochastic methods rely on the following result[7]. A region Ω

defines an area of interest for the optimization is chosen (i.e., the search space). As the most typical approach, define upper and lower bounds on each variable to be optimized such that Ω is an N-dimensional hypercube,

$$\Omega = \{\mathbf{x} : l_i \leq x_i \leq u_i, i = 1, ..., N\}$$

Defining S as a subset of Ω with a Lebesque measure of the cost function, $J : \Re^N \rightarrow \Re$ where

$$\frac{J(S)}{J(\Omega)} \geq \alpha > 0$$

Let $P(S, k)$ be the probability that one or more points of a sequence of k points, drawn randomly from a uniform distribution in Ω, lies in the subspace S, then

$$\lim_{k \rightarrow \infty} P(S, k) = 1$$

The derivation of the optimization method presented in this dissertation and the problems shown is considered a *black box* scenario where the performance, $J(\mathbf{x})$, can be readily computed, but gradients are either not available or useful and domain specific knowledge is used only within the algorithm in the black box. The cost function, $J(\mathbf{x})$, computed in the black box, is characterized by any or all of the following characteristics: non-smooth, discontinuous, ill-conditioned, nonlinear, non-convex, non-separable, multimodal, and noisy. Traditional deterministic methods suffer when the problem exhibits any of these characteristics or is of high dimensionality.

The *curse of dimensionality*, as described by Richard Bellman[10] in 1961, is associated with problems of moderate to high dimension. The problem is caused by the rapid volumetric increase associated with adding extra dimensions to a mathematical space. The classic example is to consider placing 100 points in a 1-D interval $[-1, 1]$.

To get the same spacial coverage in 10-D space, $[-1,1]^{10}$ would require 100^{10} points. A consequence of the curse of dimensionality is that an exhaustive search technique may be valuable in small dimensional spaces, but becomes unfeasible in moderate to high dimension spaces.

Definition 3. *In discrete set theory, the* cardinality *of a set is the number of elements contained in the set.*

Definition 4. *A function is* separable *if it can be optimized in a sequence of N independent 1-D optimization processes.*

Mathematically, $J(\mathbf{x})$ is separable if

$$\arg\min_{x_1,\ldots,x_N} J(x_1,\ldots,x_N) = \left(\arg\min_{x_1} J(x_1,\ldots),\ldots,\arg\min_{x_N} J(\ldots,x_N)\right) \tag{2.7}$$

A separable function is frequently made non-separable by a coordinate rotation[11]. A simple example of a separable function is the sphere model,

$$J(\mathbf{x}) = \sum_{i=1}^{N} x_i^2 \tag{2.8}$$

as each dimension may be optimized independently. For this work, when functions are separable, this fact is not exploited in the algorithm. This is done in order to make fair comparisons with other algorithms where the user may not know if separability exists, or may be exploited.

The remaining optimization review briefly describes current stochastic algorithms for optimization that are widely used in engineering (see, for example, Reference [12]). The overview provides a description of the basic workings of the algorithms, with the understanding that many modifications, by many different researchers, are made to each algorithm in order to improve a given set of performance parameters.

Stochastic algorithms all have the following advantages over deterministic optimization methods (adapted from References [3, 6, 13, 14, 15, 16, 17]):

- Wide applicability (continuity and differentiability may not be known);
- Able to handle multi-modalities, discontinuities, constraints, and noisy functions;
- No assumptions are made about the underlying problem or search space;
- Algorithms generally do not get stuck in suboptimal extrema;
- No initial or tentative solutions required (other than the definition of Ω);
- Low application and development cost (i.e., good ratio of effort to performance);
- Easily incorporated into other methods (or can be hybridized with other methods);
- Can be run interactively, allowing user-proposed solutions at any stage;
- Generally can provide multiple alternative solutions;
- Generally acknowledged as good solvers for tough problems;
- Easily parallelized for faster computation;
- Adaptability to new problems requires relatively low tailoring.

On the other hand, the stochastic algorithms described all have the following disadvantages:

- No clear stopping criteria;

- Weak or no theoretical basis;
- No guarantee of converging to the global optimal point;
- Can be computationally expensive;
- Difficult to make fair comparisons;
- Works best when user able to narrow the search space while maintaining diversity of samples;
- Difficult to predict movements of sample populations;
- Required tuning based on the problem.

The general lack of a solid theoretical basis for the evolutionary algorithms (a subset of stochastic algorithms) has affected their acceptance in some circles, although their use is widespread, as nothing better is available. With little theory, the user is left only a set of rules-of-thumb for choosing and tuning the algorithms, and in more complicated problems, the tuning is itself an optimization problem. Thus, a methodology where tuning is not required is sought.

Of all of the optimization algorithms developed to date, Fogel's *no free lunch theorem* states that there is no algorithm that is best to solve any general problem[18]. The remainder of this chapter is dedicated to an overview of several popular stochastic algorithms.

2.2 A Brief Historical Tour

The most general methods to optimize a function with a high degree of computational efficiency are gradient based methods. However, these methods require a continuous cost function where gradient information is available. Since these methods seek to ascend or descend from a starting point based on a local Taylor series approximation, we should anticipate several convergence issues. When the cost function has a large number of variables, has discontinuities, has discrete variables, or is multimodal, gradient methods are no longer the best choice of algorithm (or at best, need assistance to locate the neighborhood of the solution). For the applications, probabilistic methods are developed.

Since 1944, the science and engineering communities have known that many important and practical problems may be posed as a mathematical optimization problem[7] for sampling methods. Numerical minimization is much older, dating back to the time of Gauss or earlier. Stochastic optimization started in 1952 with the work of Robbins and Monro[19] with their stochastic approximation method. In 1966 Fogel, et. al., published evolutionary programming[20] (although the work began in 1960) which uses a simulation of the evolutionary process as a learning process. Evolutionary algorithms are a subset of stochastic algorithms where a population of sample points (of fixed size) is evolved through some processes, producing new generations of the population. Haataja says, "Evolution is a collective phenomenon of adapting to the environment and passing genes to the next generations."[21] Evolution is in and of itself not optimization. Hastings developed the Metropolis-Hastings algorithm in 1970[22], as a sampling method used to obtain a sequence of random samples from a probability density function which is generally difficult to sample from. The first genetic algorithm was proposed by Holland in 1975[23]. Kirkpatrick, et. al., proposed the simulated annealing algorithm in 1983[24] based on Hasting's work. As the algorithms are introduced in the next section, important dates and contributors are highlighted. Evolutionary algorithms mimic some observed biological process, but in general do not have a strong mathematical basis.

2.3 Overview of Popular Stochastic Algorithms

Recent solutions to numerical optimization problems in large dimensional spaces have been found using stochastic methods such as evolutionary algorithms (EAs). Stochastic algorithms have three fundamental parts: *sample*, *optimize*, and *check*[25]. Sampling involves generating random points, $\mathbf{x} \in \Omega$, and computing the associated cost value, $J(\mathbf{x})$. Optimization applies a set of rules which drives the new selection of points. Check decides if a stopping criteria is met. The check is called the most important part of the optimization algorithm by many authors, including Ref. [25]. The algorithms developed in this dissertation are assumed to be stochastic in nature, meaning the observations are placed based on generating random points. The model in each application presented is assumed to *not* be stochastic. An overview of some of the most popular methods is now be given for understanding and comparison.

In several of the approaches presented, it will be evident that they all share a common feature: namely all use some heuristic adaptation process that governs both global and local sampling to seek the extreme value of a function. In all cases, a time-varying sampling density function is implicitly invoked and it has been hypothesized [26] that it may be possible to unify many of these approaches by focusing on the underlying sample density function evolution.

2.3.1 Random Search

The Random Search technique (also known as the Monte Carlo Method) is the simplest of the algorithms in this section and lends itself to more rigorous theoretical analysis. By definition, the random search is not an evolutionary algorithm[9, 27, 7], but it is a stochastic algorithm. It is included for completeness and a comparison is made with the newly derived optimization method that is the focus of this dissertation. In the pure random search method, a specified number of points are randomly chosen (from some distribution, usually a uniform distribution) within the search space. The cost function is evaluated at each of these points and the point with the best value is considered the optimal point. The pure random search algorithm is shown in Algorithm 1.

Algorithm 1 Pure Random Search Algorithm

```
1: Set $J^* := +\infty$
2: while Not Terminate do
3:     choose $\mathbf{x} \in U(\Omega)$ as a uniform random vector in $\Omega$
4:     if $J^* > J(\mathbf{x})$ then
5:         let $J^* = J(\mathbf{x})$ and $\mathbf{x}^* = \mathbf{x}$
6:     end if
7: end while
8: (optional) begin local optimization starting from $\mathbf{x}^*$
```

The random search provides better solutions than a simple grid search[9] (which covers the search space with a specified number of equidistant points). Brookes[28] showed in 1958 that given a measure, J, then if

$$\frac{J(S)}{J(\Omega)} = \alpha \tag{2.9}$$

then the probability of finding a point in S in k points is

$$P(S,k) = 1 - (1-\alpha)^k \tag{2.10}$$

As such, the set S is made arbitrarily small at the cost of requiring a larger number of points.

Equation (2.10) shows that a large number of points are required to reach a small region with high probability around the global minimum. For this reason, many combinations of the Random Search method and deterministic methods have been proposed. Two modifications are shown in Algorithms 2 and 3 below [7]. The Multistart algorithm is very popular in the literature[7].

Algorithm 2 Multistart Random Search Algorithm

```
1: while Not Terminate do
2:    Select a random $\mathbf{x} \in \Omega$
3:    Start a local minimization algorithm from $\mathbf{x}$ with given stopping criteria
4:    if $\mathbf{x}$ is probably a global minimum then
5:       STOP
6:    else
7:       Return to Step 1
8:    end if
9: end while
```

Algorithm 3 Hartman's S2 Algorithm

```
1: Set $v = \infty$
2: while Not Terminate do
3:    Select a random $\mathbf{x} \in \Omega$
4:    if $J(\mathbf{x}) > v$ then
5:       Return to Step 2
6:    else
7:       Start local minimization from $\mathbf{x}$ to $M_j$
8:       Set $v = J(M_j)$
9:       Return to Step 2
10:   end if
11: end while
```

One drawback of the simple random searches presented here is that no cost function information is used to adaptively guide the searches. In other words, the method has no memory and does not learn from previous experience. Each point is chosen independently from the previous set of points.

2.3.2 Genetic Algorithm

The Genetic Algorithm (GA) is a widely used and well known algorithm that has been a staple in numerical optimization problems for three decades[23, 29] due to its general ease of use and solving power. The GA is a part of evolutionary computing, which mimics, to some degree, the biological evolutionary process as described by Darwin. As such, the general terminology is taken from biology. The variables to be optimized are mapped into a *genome* which is then evolved throughout the process of optimizing the problem. The GA randomly creates a population consisting of individuals, computes the cost associated with each individual, selects individual elements to reproduce, performs *crossover* and *mutation* operations, and creates *offspring* which replace the worst ranked portion of the population. This process repeats until the convergence criteria is met. Algorithm 4 shows pseudocode for the GA.

Algorithm 4 Genetic Algorithm Pseudocode

```
1: Initialize $\mathbf{x}_i$ for $i = 1, ..., m$
2: Compute $J(\mathbf{x}_i)$ for $i = 1, ..., m$
3: while Not Terminate do
4:     Select individual members for reproduction based on best fitness values
5:     Generate new individuals (i.e., offspring) through crossover and mutation operations
6:     Compute $J(\mathbf{x}_j)$ for the offspring
7:     Replace individuals with worst fitness values with offspring
8:     if Converged then
9:         STOP
10:    end if
11: end while
```

The collection of independent variables, $\mathbf{x}$, of the problem are called *chromosomes* and a given single variable is called a gene[30]. The crossover operation takes the first section of a chromosome from one parent and the second section of a chromosome from the other parent to reproduce an offspring. The probability of crossover is typically between 0.6 and 0.8. The mutation operation selects one gene in a chromosome and changes the value of that gene. The probability of mutation is generally between 0.1 and 0.2. These operations keep the offspring from being exact copies of the parents. Combinations of genes in the population may survive to the next generation even if the combination is not optimal because the process is based on probability[21]. Obviously, genome, chromosome, parents, offspring, crossover, and mutation are heuristic labels used to describe the adaptive learning process. These heuristic labels have appeal but there is no underlying biological or mathematical proof of their uniqueness or optimality.

For a real-coded GA (as opposed to binary coded), the crossover is applied to the individual variables according to the following probability distribution

$$P(\beta) = \begin{cases} 0.5(\eta+1)\beta^{\eta} & \text{for } \beta \leq 1 \\ 0.5(\eta+1)/\beta^{\eta+2} & \text{otherwise} \end{cases} \tag{2.11}$$

with $\eta = 2$ being suggested as η controls how close the parent and offspring solutions are. However, most implementations (MATLAB included) do not allow this parameter η to be tuned.

Much discussion has arisen about the role of mutation and crossover[13]. The mutation plays more of a role of creating diversity (global search) in the population while crossover tends to exploit good solutions to find better solutions in a neighborhood of the good solutions found already (local search). General strategies in

GAs (and other EAs) include having the role of mutation diminish and the role of crossover increase as the number of generations grows. This ensures a diverse population at the beginning and an exploitation of the good solutions towards the end of the run.

Of the algorithms described in this section, the GA is the most widely, and successfully, applied to problems in engineering. Researchers have found that small changes in the crossover fraction or mutation rate lead to significantly different answers on many problems, thus making tuning the GA both difficult and necessary. For some problems, finding the proper values of the tuning parameters becomes itself an optimization problem[14]. The crossover and mutation operators are heuristically and operationally well described but not generally well understood from a mathematical standpoint.

2.3.3 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm is a stochastic method which is inspired by the social behavior and movements of birds and fish in formation and takes advantage of information sharing among the swarm, called "collective intelligence". The PSO algorithm has received considerable attention recently[31, 32, 33, 34, 35, 36, 37]. Like the GA, the PSO is a population-based algorithm. Each element in the population set has a *position* and *velocity* in the search space. In this sense, the position represents the N-D state value of a point in the search space, with an associated cost function value and the velocity determines the position update. Every individual also has a memory of their personal best position (i.e., best cost function value) as well as the best global position (including current global best cost function value). The algorithm moves the particles based on a combination of best global and individual positions as well as the velocity of an individual. In this way, the elements (or particles) swarm to the global optimal location. It should be noted that the PSO requires at least two particles to function, but the power is in the social interaction of a large number of particles.

The PSO has three parameters to tune where each is proportional to: an individual's previous velocity; the personal best position; and the global best position. Varying these values drastically impacts the convergence speed and accuracy.

The PSO algorithm pseudocode is shown in Algorithm 5. Let $\mathbf{x} \in \Re^N$ and $\mathbf{v} \in \Re^N$. Define $\hat{\mathbf{x}}_i$ as the best position for the i^{th} particle and $\hat{\mathbf{g}}$ be the global best position (with *best position* defined as the position that gives the minimum cost function value). Finally, let m be the number of particles in the swarm. In the

Algorithm 5 Particle Swarm Optimization Algorithm Pseudocode

```
1: Initialize $\mathbf{x}_i$ and $\mathbf{v}_i$ for $i = 1, ..., m$
2: $\hat{\mathbf{x}}_i \leftarrow \mathbf{x}_i$ and $\hat{\mathbf{g}} = \min_{\mathbf{x}_i} J(\mathbf{x}_i)$ for $i = 1, ..., m$
3: while Not Terminate do
4:     for $i = 1$ to $m$ do
5:         Randomly generate $r_1, r_2 \in U[0, 1]$ for $j = 1, ..., m$
6:         Update particle velocities: $\mathbf{v}_i \leftarrow \omega\mathbf{v}_i + c_1 r_1(\hat{\mathbf{x}}_i - \mathbf{x}_i) + c_2 r_2(\hat{\mathbf{g}} - \mathbf{x}_i)$
7:         Update particle positions: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$
8:         Update particle best positions:
9:         if $J(\mathbf{x}_i) < J(\hat{\mathbf{x}}_i)$ then
10:            $\hat{\mathbf{x}}_i \leftarrow \mathbf{x}_i$
11:        end if
12:        Update global best position:
13:        if $J(\mathbf{x}_i) < J(\hat{\mathbf{g}}_i)$ then
14:            $\hat{\mathbf{g}}_i \leftarrow \mathbf{x}_i$
15:        end if
16:    end for
17: end while
```

algorithm, the value ω is the *inertial constant*. Generally, this value is slightly less than 1, but it can be a random value for each particle. The random values c_1 and c_2 are the *cognitive* and *social* components, respectively. These values determine the influence on the movement of particles based on the particle's personal best position, c_1, and the global best position, c_2. Typical values of c_1 and c_2 are very near 2.

2.3.4 Differential Evolution

Differential Evolution (DE) is an algorithm based on a parallel direct search method in which each element of the population is updated by a weighted difference of two or more randomly selected population elements[38, 39, 40]. This updated value replaces the old element when it has a better cost function value (i.e., $J(\mathbf{x})$ is lower). The DE algorithm has found recent use in interplanetary mission design[39, 41]. Differential Evolution requires a non-zero crossover probability and strategy in order to ensure the algorithm does not get stuck in local extrema, however many practical variants of the DE exist based on varied crossover strategies.

The DE uses a mutation weight, F, ranging between 0 and 2, which serves to amplify the difference between the trial vectors. The crossover ratio is set by the user and bounded by $[0, 1]$. DE requires a population greater than 4 in order to begin iteration. For each dimension of a trial vector, if a random number (in $[0, 1]$) is less than the crossover ratio, the value of the mutant vector becomes the value of the trial vector. If the random number is greater than the crossover ratio, then the trial vector maintains its value. Reference [40] suggests six variants of the DE as listed below as DE/x/y. To interpret these values, x specifies the vector to be mutated (either random, best cost function value, or rand-to-best meaning the perturbation is placed between a random vector and the vector with the best cost function value) and y specifies the number of difference vectors to be used. Further work[38] added a crossover scheme option of binary and exponential.

1. DE/best/1:

$$y = x_{best} + F(x_1 - x_2)$$

2. DE/rand/1:

$$y = x_1 + F(x_2 - x_3)$$

3. DE/rand-to-best/2:

$$y = x_1 + Fx_{best} + x_2 - x_3)$$

4. DE/best/2:

$$y = x_{best} + F(x_1 + x_2 - x_3 - x_4)$$

5. DE/rand/2:

$$y = x_5 + F(x_1 + x_2 - x_3 - x_4)$$

6. DE/rand-to-best/1:

$$y = x_1 + G(x_{best} - x_1) + F(x_2 - x_3)$$

where F and G are weights, x_i are the randomly chosen N-dimensional values for $j = 1, ..., 5$, x_{best} is the state value of the individual in the population with the best cost function value. The value of f is generally randomly chosen[39] as $f \in U[-1, 1]$ and the crossover probability is generally 0.8[40]. Price suggests a highly beneficial method, and thus the most widely used, is DE/rand/1 with a binary crossover scheme[40].

A pseudo-code for the Differential Evolution scheme is presented in Algorithm 6.

Algorithm 6 Differential Evolution Algorithm Pseudocode

```
1: Initialize population $\mathbf{x}_i$ and compute cost value $J(\mathbf{x})$
2: while Not Terminate do
3:     for $i = 1$ to Number of Iterations do
4:         Randomly select parents, $\mathbf{p}_i$, for $i = 1, 2, 3$
5:         Create initial candidate combining genes from the parents
6:         Create final candidate, $\mathbf{c}_i$, by crossing over genes from parents and initial
           candidate
7:         Evaluate the associated cost of the candidate, $\mathbf{c}_i$
8:         if $J(\mathbf{c}_i) < J(\mathbf{p}_i)$ then
9:             $\mathbf{p}'_i = \mathbf{c}_i$
10:        else
11:            $\mathbf{p}'_i = \mathbf{p}_i$
12:        end if
13:        $\mathbf{p}_i = \mathbf{p}'_i$
14:    end for
15: end while
```

2.3.5 Simulated Annealing

The Simulated Annealing[24, 42, 43, 44] (SA) algorithm is a variant of the Monte Carlo technique that is based on annealing from metallurgy. Annealing is a technique of heating a material until it melts and then gradually cooling the material at a controlled rate in order to form a perfect lattice of particles, increase the size of the crystals, and reduce the crystalline defects. This final state is a minimum energy configuration of the solid. The SA algorithm is of course based on a heuristic analogy to annealing. The method seeks to drive the physical system to a minimum energy configuration by carrying out the "cooling" slowly. For a given temperature, T, the probability of a material being in a state w is defined Boltzmann's distribution

$$P(w) \propto e^{-E(w)/k_b T} \tag{2.12}$$

where $E(w)$ is the *energy* of the state w and k_b is Boltzmann's constant. The state with the highest probability at equilibrium is the state with the lowest energy.

The algorithm picks a neighbor of the current point and computes the energy (i.e., cost function value) at the chosen point. The neighboring point is accepted if the cost function value is improved. The SA algorithm avoids becoming trapped in local minima by probabilistically accepting state transitions which correspond to a deterioration in the cost function value (i.e., accepting an uphill move) according to a Metropolis criteria[45] and the Boltzmann distribution. As the optimization process continues, the probability of accepting an inferior point goes to zero. This is called the annealing schedule and is an input parameter by the user. The pseudocode in Algorithm 7 shows the flow of the SA algorithm.

It should first be noted that the Metropolis criterion for accepting a new configuration, $\mathbf{x_2}$, in place of the current configuration, $\mathbf{x_1}$, is

$$\beta_T(\mathbf{x_1}, \mathbf{x_2}) = \min\left(1, e^{-[J(\mathbf{x_1})-J(\mathbf{x_2})]/T}\right) \tag{2.13}$$

The SA algorithm is not population based but instead generates a sequence of directional searches. One main drawback to the SA algorithm is that it (in the most basic form) requires the cost function to be continuous in the search space. However, the SA is successfully used in discrete optimization problems and solved the Traveling Salesman Problem for 400 cities[24].

The choice of initial temperature, T_0, is critically important to the SA algorithm. A value too high considerably slows the computational speed while a value too close to 0 excludes a global search in favor of a local search. The temperature decrement strategy is also an important factor. If the temperature decreases too quickly, the algorithm risks becoming trapped in a poor local minimum. However if the algorithm

Algorithm 7 Simulated Annealing (Hide-and-Seek) Algorithm Pseudocode[43]

```
1: while Not Terminate do
2:    Choose a starting point, $\mathbf{x_0} \in \Omega$
3:    Choose a high enough starting temperature, $T_0 > 0$
4:    while $T \neq 0$ do
5:       Choose a search direction $\mathbf{g}_j \in \Omega$ with a uniform distribution
6:       Choose a step size, $\lambda_j$, from a uniform distribution such that $\bar{\mathbf{x}}_j = \mathbf{x_j} + \lambda_j \mathbf{g}_j \in \Omega$
7:       Choose $c_j \in U[0,1]$
8:       Determine the next search point $\mathbf{x_{j+1}}$ from
             $$\mathbf{x_{j+1}} = \begin{cases} \bar{\mathbf{x}}_j & \text{if } c_j \in [0, \beta_T(\mathbf{x_j}, \bar{\mathbf{x}}_j)] \\ \mathbf{x_j} & \text{if } c_j \in [\beta_T(\mathbf{x_j}, \bar{\mathbf{x}}_j), 1] \end{cases} \quad (2.14)$$
9:       if $J(\mathbf{x_{j+1}})$ is less than all previous function values then
10:         Update temperature, $T$
11:      else
12:         Go back to Step 2
13:      end if
14:   end while
15: end while
```

decreases to slowly, the computational burden greatly increases. Finally, the method of choosing an appropriate neighborhood for the next search direction and point has not been well studied[25].

2.3.6 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDA) is a general term that covers a class of many different optimization methodologies. For simplicity, the general form of the EDA is considered here. In it's simplest form, the EDA[15, 46] is a population based method that generates a population, evaluates the cost function value associated with each member in the population, and uses selected individuals to estimate a probability distribution from which the next generation is sampled. In this way, the crossover and mutation operators are not necessary and the relationships between variables representing the individuals are explicitly expressed through the probability distribution. The difficulty of tuning the parameters[14], such as crossover and mutation, along with the fact that the movement of a population is very difficult to predict in the search space[15] have motivated the development of EDAs. EDAs benefit from having some theoretical backing [47].

For EDAs, the computational bottleneck is in computing the probability density function[15] after a population is generated. One drawback of the EDA is choosing the basis for mathematically describing the probability density function (this is most simply a curve fitting exercise, but generally is an expression of the joint probability distribution where every variable is considered independent of the rest) and choosing which points in the database to use. Algorithm 8 shows a pseudocode for a general EDA.

Algorithm 8 Estimation of Distribution Algorithm[15]

```
1: Generate at random $\mathbf{x}_j \in \Omega$ for $j = 1, ..., m$
2: Evaluate the cost for all $\mathbf{x}_j$
3: while Not Terminate do
4:     Select $k \leq m$ individuals from $\mathbf{x}_j$ based on a given selection method
5:     Estimate the probability distribution based on the selected sample of $k$ points
6:     Sample the probability distribution to obtain a new $\mathbf{x}_j$ for $j = 1, ..., m$
7: end while
```

The EDA is not used in comparisons in any of the applications, but is here shown because of it's similarity to the optimization algorithm developed in the next chapter which is the focus of this dissertation. The reader will see specific differences as the algorithm is developed.

2.4 Brief Review of Astrodynamics

The remainder of this chapter is meant to serve as a brief introduction to astrodynamics. This discussion is deemed to be necessary in order to understand the applications presented in the dissertation. Many excellent texts exist that allow for a deeper study, including those by Battin [48], Vallado [49] and Schaub and Junkins [50], just to name a few that are readily available.

The most relevant equations for the applications presented are those concerned with Keplerian two body motion. Although a significant amount of research has been devoted to solving the orbit problem with perturbations, such as J_2, the main term of the spherical harmonic expansion of the Earth's gravitational field, the problems solved in the next chapter are not concerned with any perturbations.

Keplerian two body motion is the approximation of satellite orbits with conic sections (i.e., straight line, circle, ellipse, parabola, and hyperbola). The central gravitational body occupies on of the foci.[1] The conic section approximation is accurate enough to study the general characteristics of satellite motion. For a more accurate analysis of satellite motions, higher order perturbations must be included which make the analysis expensive to perform.

Keplerian motion is governed by Kepler's three laws of planetary motion which approximate the motion of the planets around the Sun:

1. The planets orbit the Sun in an ellipse with the Sun at one of the foci.
2. The radius vector from the Sun to each planet sweeps out equal areas during equal intervals of time.
3. The square of the orbital period of a planet is proportional to the cube of the

[1]The Latin word *focus* translates as *fire* as the Sun occupies one focus of the planets' orbits.

semimajor axis of its orbital ellipse.

These laws were discovered empirically by Johannes Kepler (1571-1630) around 1605 and published over a span of subsequent years. Kepler derived the laws using observational data of Mars taken by Tycho Brahe (1546-1601).

Roughly a century after Kepler's discoveries, Sir Isaac Newton (1643-1727) derived his law of universal gravitation and law of motion[2]. From Kepler's second law, it follows that the force (acceleration in this case) is directed towards the Sun. From Kepler's second and third laws, it follows that the magnitude of the force is inversely proportional to the square of the distance to the Sun. These two facts suggest that the Sun is the physical cause of the force (i.e., acceleration) on the planets. Newton's law of universal gravitation is in fact remarkably simple. Following the definition of force as mass multiplied by acceleration, Newton derived

$$\mathbf{F} = \frac{-Gm_1m_2}{r^3}\mathbf{r} \tag{2.15}$$

where $G = 6.67428 \cdot 10^{-11}$ N-m/kg^2 is the universal gravitational constant, m_1 and m_2 are the masses of the two spherically symmetric bodies, and $\mathbf{r}$ is the vector from the center of mass of body 1 to the center of mass of body 2, and $\mathbf{F}$ is the gravitational force between the two bodies. Newton's law of motion is then derived as

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} \tag{2.16}$$

where $\mu = G(m_1 + m_2)$ is the planetary gravitational constant. For spacecraft orbiting the Earth, $m_2 \ll m_1$ and the the approximation $\mu = Gm_1$ is used, which is independent of the spacecraft mass. For the Earth, $\mu = 398600.441$ km^3/s^2.

[2]Newton published *Philosophiae Naturalis Principia Mathematica* on July 5, 1687.

Newton generalized Kepler's laws by assuming that all bodies with mass in the solar system attract each other through the gravitational force and this force is directly proportional to the masses and distance between the bodies. As the mass of the Sun is much larger than the mass of any of the planets (or the planets combined for that matter), Kepler's model well approximates the motion.

The two body problem is valid for the developments and applications in this dissertation as the following assumptions are made. First, the motion of one body is studied (as the mass of the spacecraft is much smaller than the mass of the Earth). In addition, the motion of the spacecraft is assumed to be within the *sphere of influence* of a massive body, meaning that interference of other gravitational bodies and other disturbances (e.g., atmospheric drag or solar pressure) do not greatly influence the spacecraft's motion. Within the sphere of influence of the planet, the two body approximation is remarkably accurate. For the Earth, the sphere of influence is approximately $R_{\text{SOI}} = 1.5 \cdot 10^6$ km $= 0.01$ AU.[3]

2.4.1 Conservation of Angular Momentum

By taking the cross product of Eq. (2.16) with the position vector, the conservation of angular momentum is easily proven as follows

$$\dot{\mathbf{h}} = \mathbf{r} \times \ddot{\mathbf{r}} = \frac{d}{dt}(\mathbf{r} \times \dot{\mathbf{r}}) = 0 \tag{2.17}$$

This states the angular momentum per unit mass, $\mathbf{h}$, is constant (when no external forces are acting on the body). Thus, the motion occurs on a plane defined by $\mathbf{r}$ and $\dot{\mathbf{r}}$, the position and velocity vectors, respectively. In polar coordinates this is written

[3]1 AU or *astronomical unit* is the mean distance between the Sun and the Earth, 149,598,000 km.

as

$$h = r^2\dot{\varphi}\hat{i}_h \tag{2.18}$$

where $\hat{i}_h$ is the vector perpendicular to the $\mathbf{r}$, $\dot{\mathbf{r}}$ plane. Note that Kepler's second law is proven in that the rate at which the radius vector sweeps out area is $1/2r^2\dot{\varphi}$.

2.4.2 Energy Integral

Beginning with the eccentricity vector, which is a constant of the orbital motion,

$$\mu\mathbf{e} = \dot{\mathbf{r}} \times \mathbf{h} - \frac{\mu}{r}\mathbf{r} \tag{2.19}$$

the orbit energy can be derived. It should be noted that $\mathbf{e}$ is always directed at the periapsis and the magnitude, e is a constant known as the orbit eccentricity. Investigating the square of the magnitude of 2.19,

$$||\mathbf{e}||^2 = \mathbf{e}\cdot\mathbf{e} = 1 - e^2 = \frac{h^2}{\mu}\left(\frac{2}{r} - \frac{v^2}{\mu}\right) \tag{2.20}$$

From the geometry of conic sections, the semilatus rectum is defined as $p = h^2/\mu$ and the semi-major axis as

$$a = \frac{1}{\alpha} = \left(\frac{2}{r} - \frac{v^2}{\mu}\right)^{-1}$$

which also gives $p = a(1-e^2)$. The sum of the kinetic and potential energy per unit mass is

$$E = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a} \tag{2.21}$$

which is in fact a constant of the motion.

2.4.3 Conic Equation in Polar Form

By taking the dot product of the position and eccentricity vectors

$$\mathbf{r} \cdot \mathbf{e} = \frac{h^2}{\mu} - r \tag{2.22}$$

$$r = \frac{p}{1 + e\cos(\varphi)} \tag{2.23}$$

where φ is defined as the *true anomaly* which is defined as the angle between the eccentricity vector and the radius vector. The orbit is defined by a conic depending on the value of e as shown in Table 2.1.

Table 2.1: Value of e Defining Conic Sections

e Value	Conic Section
e=0	Circle
$0 < e < 1$	Ellipse
e=1	Parabola
$e > 1$	Hyperbola

These conic sections are visualized by imagining the intersection of a plane with a cone at different angles. For elliptical orbits (of which circular are a special case), the semi-major axis, a, is positive. For hyperbolic orbits, the semi-major axis, a, is negative, and for parabolic orbits $a \to \infty$.

2.4.4 Orbital Elements

The orbit of a spacecraft is completely determined if at any given time, t, the three-dimensional position and three-dimensional velocity are known. Thus, six parameters are required to completely identify the state of the orbit at any given time. The choice of the parameters is not unique, however. The classical orbital elements are the most well known set defining the orbit through:

- semi-major axis, a
- eccentricity, e
- inclination, i
- right ascension of the ascending node, Ω
- argument of perigee, ω
- mean anomaly (related to true anomaly through Kepler's equation), M

The first two parameters define the shape of the orbit. The orientation of the orbit plane is defined by the third through fifth parameters. The sixth element defines the position of the spacecraft on the orbit and is a time parameter. The transformation between orbital elements and and Earth-centered inertial Cartesian frame is well documented[49, 50]

2.4.5 Lambert's Problem

Lambert's problem is the time constrained, two-point boundary value problem for Eq. (2.16):

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r}$$

The problem statement is: given two positions at two times, $\mathbf{r}(t_1)$ and $\mathbf{r}(t_2)$, find the solution for $\mathbf{r}(t)$ that satisfies Eq. (2.16). The position and velocity as a function of time are then known, and thus, the orbit is completely defined for all times. Many solution methods exist to solve Lambert's problem[48, 51]. Battin's method is used in the examples shown in this dissertation.

2.4.6 Definition of Δv

The change in velocity required to move between two orbits is called Δv. To be complete, a velocity change could be imparted that moved a spacecraft between two points in the same orbit, but this case is not considered in this dissertation. Equation (2.21) allows

$$v^2 = \mu \left(\frac{2}{r} - \frac{1}{a} \right) \tag{2.24}$$

and thus at any point in the orbit the velocity can easily be computed. If two points in two different orbits are given, the required velocity change can be computed. Δv values are used as a general description as fuel consumption requires specific metrics of an engine to be known. Thus, Δv is a more general term dictating the energy requirement to move between two orbits.

The value of Δv is generally given as the modulus of the corresponding change in the velocity vector, but for the applications in Chapter IV, the modulus and direction is investigated. In the case where the vector values are required, the change in velocity is given as $\Delta \mathbf{v}$, where each component is listed.

3. LEARNING APPROACH THEORY AND EXAMPLES

This chapter provides the theory and several numerical examples of benchmark problems to validate the *Learning Approach to Sampling Optimization* method. The Learning Approach (LA) is based on *rejection sampling* (also called the accept-reject method)[52]. This new stochastic optimization method will be demonstrated to converge rapidly to an accurate solution and to more thoroughly search the proper extrema (minima or maxima) across the entire space than existing methods. A qualitative motivation is to use both global and local information learned by the algorithm to recursively *shape* the sampling density function such that the "best" regions identified are sampled more densely.

As evolutionary algorithms (EAs), a particular subset of stochastic algorithms, have been shown to outperform classical deterministic optimization techniques[53], the LA will be compared to the widely applied Genetic Algorithm (GA) on several benchmark problems in multiple dimensions (although purely for comparison in this chapter, as the next chapter introduces applications in astrodynamics and more rigorous comparisons are made).

This chapter introduces rejection sampling followed by the LA theory and development. Then several benchmark problems are solved with comparisons where appropriate.

3.1 Rejection Sampling

Rejection sampling, as proposed by John von Neumann in 1951[52], is typically used to generate a random set of observations from a distribution and is of particular use when the form of the probability density function (PDF) makes sampling difficult. If a target density, $f(\mathbf{x}|y)$, is difficult to sample from, one can sample from a density function that is an integral envelope function, $f(\mathbf{x}|y) < M\,G(\mathbf{x}|y)$ with $M > 1$, much more quickly. To get a random draw from the density $f(\mathbf{x}|y)$, simply generate a sample from within $G(\mathbf{x}|y)$ and accept or reject it according to an acceptance probability, $\alpha(\mathbf{x}|y)$. In his paper, von Neumann[52] showed that the choice

$$\alpha(\mathbf{x}|y) = \frac{f(\mathbf{x}|y)}{G(\mathbf{x}|y)} \tag{3.1}$$

will correctly produce Independent Identically Distributed (IID) samples from the sample space $f(\mathbf{x}|y)$.

The rejection sampling algorithm as used in the rest of this dissertation assumes that appropriate bounds on the distribution are known–meaning that the function $G(\mathbf{x})$ is known. (Methods of bounding the function are discussed in this chapter.)

The rejection sampling algorithm is, in-and-of-itself, very simple. The problem starts with a given function to be sampled from, $f(\mathbf{x})$, with bounds on $\mathbf{x}$ and an expected bound on $f(\mathbf{x})$, and provides a randomly distributed sample set from the distribution $f(\mathbf{x})$. The rejection sampling algorithm then follows in the pseudocode shown in Algorithm 9.

Algorithm 9 Rejection Sampling Algorithm

```
1: while Not Terminate do
2:     Randomly choose (within the given bounds) $\mathbf{x} \in \Re^N$ and $y \in \Re$
3:     if $y \leq f(\mathbf{x})$ then
4:         Accept $\mathbf{x}$
5:     else
6:         Reject $\mathbf{x}$
7:     end if
8: end while
```

Algorithm 9 will be shown again as the backbone of the optimization algorithm being developed in this chapter. Rejection sampling is the tool which allows the selection of points in a novel way inside of the optimization technique.

Rejection sampling itself provides only a uniformly, randomly distributed set of points bounded by $f(\mathbf{x})$, and thus in the purest form cannot be used for optimization. However, it will be shown to be extremely powerful in the selection of points inside of an optimization algorithm. Figures 3.1 and 3.2 show a simple example of a two-dimensional reconstruction of the continuous function

$$f(x,y) = \left| \cos(x^2)\,\sin(y^3) + \frac{x\,y}{10} \right|$$

Figure 3.1 is the prescribed 2-D function and Figure 3.2 is the reconstructed 2-D histogram by the rejection sampling method.

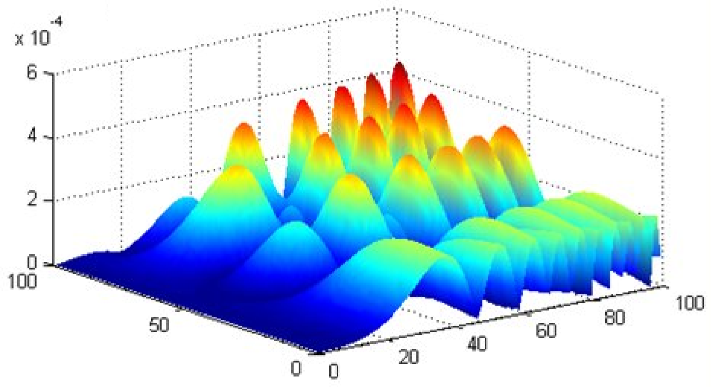

Figure 3.1: Prescribed 2-D PDF

The rejection sampling approach is used in many different applications. Just to highlight one (discrete) example, the top plot of Figure 3.3 represents a portion of the background noise histogram of a camera with a broken bit in the analog-to-

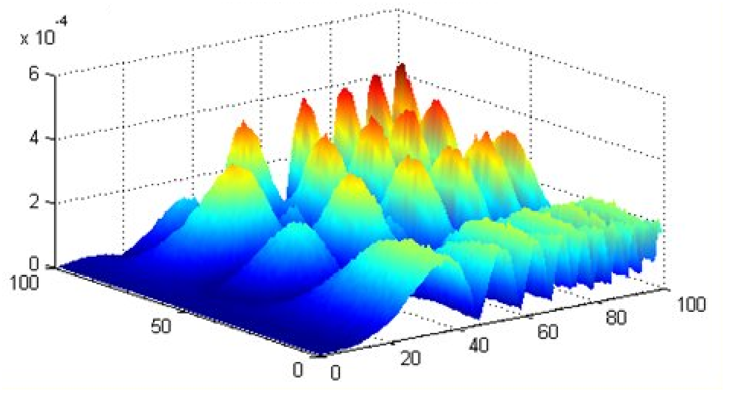

Figure 3.2: Simulated 2-D PDF (from Histogram)

digital converter readout register. The rejection sampling reconstruction is given in the bottom plot. The reconstructed PDF allowed an accurate simulation of the camera's noise characteristics based on an actual imagery. This example is just a simple application to show the power of rejection sampling in a real-world situation.

3.2 Learning Approach to Sampling Optimization: Theory

The ideal optimization approach is able to:

- adapt to a wide class of possible applications
- provide consistent and reliable convergence
- able to locate multiple satisfactory near-optimal points (including the global optimal)
- easy to implement

Based on the characteristics of the ideal optimization approach, a scheme to a) determine state values, $\mathbf{x}$, that optimize the cost function and to b) intelligently weight the random selection of points toward the optimal solution of the cost function

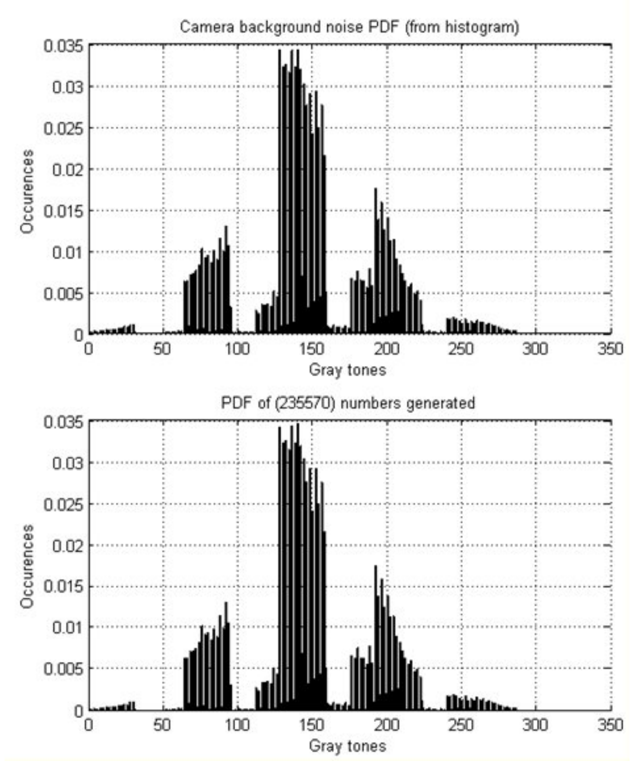

Figure 3.3: Example of prescribed and simulated 1-D discrete PDF.

is required in order to rapidly converge to the desired extrema is sought. A modified rejection sampling scheme is implemented to sample from the given cost function, $J(\mathbf{x})$. While this is not a purely random selection of points, it allows a pseudo-random selection, weighting the selection of points toward the extrema but also allowing the entire search space to be explored with some probability. This is the first place where the LA algorithm deviates from any random (including adaptive random) search techniques.

The claim of weighting the selection of points toward the extreme values is based on the rejection-sampling method itself, which probabilistically accepts points based on the PDF function. It is shown that the PDF is approximated by the cost function, increasing the probability of accepting an extreme point.

Previous versions of the LA algorithm[54, 55], present improvements to increase computational speed and make memory usage more efficient in the computer implementation. The final version of the algorithm is presented here. The backbone of the algorithm is still rejection sampling [52] (as shown in Algorithm 9), and the flow of the N-dimensional Learning Approach to Sampling Optimization is presented in Algorithm 10.

Algorithm 10 Learning Approach to Sampling Optimization Algorithm for Minimization

```
1: Select a number of random points of $\mathbf{x} \in \Omega$ ($2N$–twice the dimension–random
   points were used unless otherwise specified)
2: Compute the associated cost function, $J(\mathbf{x})$, for each of the selected points
3: while Not Terminate do
4:     Randomly choose a point $(\mathbf{x}_{\text{ran}}, y)$ where $\mathbf{x}_{\text{ran}} \in \Omega$ and $y \in J(\mathbf{x})$
5:     Identify the nearest neighbor to $\mathbf{x}_{\text{ran}}$, call this point $\bar{\mathbf{x}}$
6:     if $y \geq J(\bar{\mathbf{x}})$ then
7:         Add $\mathbf{x}_{\text{ran}}$ to the sample database (i.e., accept $\mathbf{x}_{\text{ran}}$)
8:     else
9:         Return to Step 4 (i.e., reject $\mathbf{x}_{\text{ran}}$)
10:    end if
11: end while
```

The algorithm starts by distributing some number of points, $\mathbf{x}$, throughout the search space, Ω. These points need not be uniformly distributed, especially if some *a priori* information exists about the cost function. Only one point need be chosen to start the algorithm, but for all of the examples shown in this dissertation, the initial number of points was taken as $2N$, unless otherwise specified. The associated cost function, $J(\mathbf{x})$, is computed for each point. These steps initialize the LA by building an initial database. The points in the database serve to approximate the cost function and are used in the rejection-sampling acceptance criteria.

Once the initial database is constructed, the optimization loop starts. Until the termination condition is met, the following steps are taken. A single N-dimensional random value of $\mathbf{x}$ is chosen and single 1-dimensional random value of y is chosen, where y is bounded by the cost function (methods of bounding the cost function are discussed in Section 3.6). The nearest neighbor to the vector $\mathbf{x} \in \Omega$ is determined and labeled $\bar{\mathbf{x}}$. The rejection sampling method is then employed to determine if the point satisfies the criteria for being added to the database. For minimization, $\mathbf{x}$ is kept if $y \geq J(\bar{\mathbf{x}})$, while for maximization, $\mathbf{x}$ is kept if $y \leq J(\bar{\mathbf{x}})$. If $\mathbf{x}$ is rejected, then a new pair $(\mathbf{x}, y)$ is chosen. If $\mathbf{x}$ is accepted, then the value y is discarded and the cost function, $J(\mathbf{x})$, is evaluated at the accepted value of $\mathbf{x}$. The data points $\mathbf{x}$ and $J(\mathbf{x})$ are then added to the database of points. Then another pair $(\mathbf{x}, y)$ is chosen. This process continues until the stopping criteria is met, which consists of a number of total points, function calls, or accuracy measure met.

As the random points are accepted based on the relation of y with respect to $J(\bar{\mathbf{x}})$, the selection of points are weighted more toward the appropriate extrema– this is better understood through the following examples. However, since the cost function is evaluated at every accepted value of $\mathbf{x}$ (and the random choice of y is discarded), the entire cost function is explored with the proper extrema (i.e., maxima or minima) being more heavily sampled. Thus, the criteria for accepting or rejecting the randomly chosen point is approximately the cost function itself as more points are accepted.

As can be seen from Algorithm 10, the technique is implemented in only a few lines of code. The fact that only accepted points require a cost function evaluation makes the algorithm very fast in terms of computation (assuming that the cost function evaluation requires significantly more operations than generating a random number). Also it is important to highlight that *there are no tuning parameters* as in the algorithms reviewed in Chapter 2. While the algorithm is fairly simple, it is shown to be extremely powerful and robust.

3.3 A Simple Example

The following example provides a simple problem to illustrate the LA algorithm. The objective is to minimize a multi-minima, 1-D function, and the results are shown in Figures 3.4 through 3.6. The cost function for this example is taken from Reference [9] as a "difficult" 1-D test function that represents a practical problem. The function is written as

$$J(x) = -\sum_{k=1}^{5} \sin((k+1)x + k) \tag{3.2}$$

The bounds are $-10 \leq x \leq 10$ and roughly $-4 \leq J(x) \leq 5$. As the function is periodic with period of 2π, there are twenty total minima, three of which are equivalent global minima in the range.

Figure 3.4 shows the cost function $J(x)$ and the first ten points (of which two were randomly chosen to start the algorithm). Figure 3.5 shows the first 100 accepted points. Note that in Figure 3.6, after only 200 points, most of the minima (local and global) are well explored with a large number of points near each of the global minima.

For this example, the three global minima are located at $(x, y) = (-6.7211, -3.3729)$, $(-0.4369, -3.3729)$, and $(5.8463, -3.3729)$. The errors achieved for this particular run, where only 200 points are accepted, are less than 0.01%. However, these results cannot be guaranteed to be repeatable as the algorithm still contains a random selection of points (this is only an example to illustrate the algorithm, statistics of multiple runs are given later).

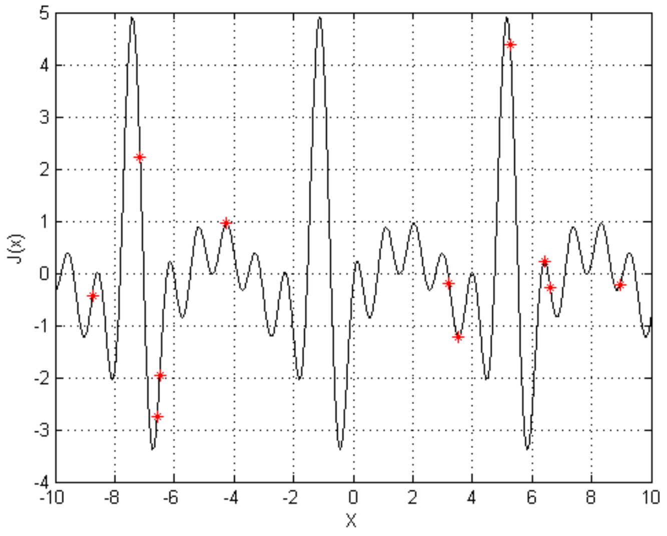

Figure 3.4: Simple Example: Initial 10 Samples

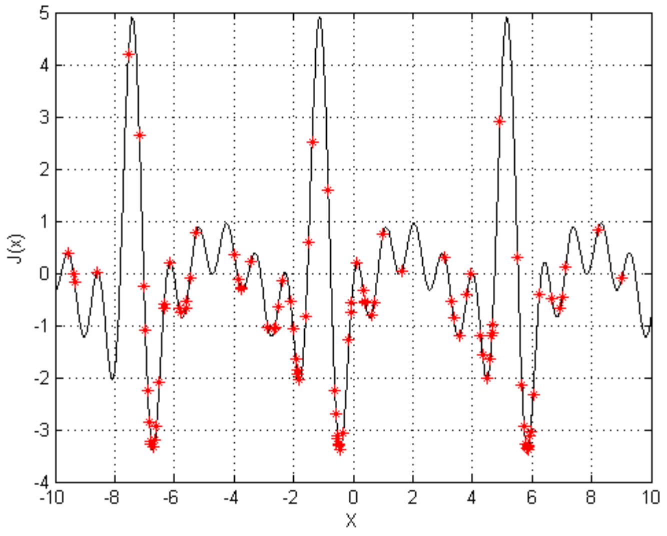

Figure 3.5: Simple Example: 100 Points

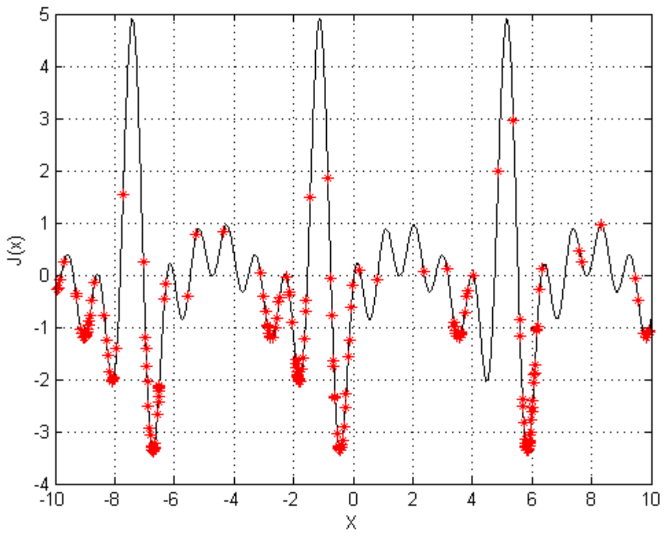

Figure 3.6: Simple Example: 200 Points

Table 3.1 shows the number of function evaluations for minimizing Eqn. (3.2) using a series of deterministic (Pijav and Batish) and statistical (Žil 1 & 2, Strong, and Brent) algorithms, given the necessary derivatives (to second order), bounds on all of the derivatives given, and the Lipschitz constant. The algorithms are thoroughly described in Ref. [9] and the table below is a modification of Table 8.4 in Ref. [9]. Törn noted that if the estimates of the Lipschitz parameter or other limiting values are too high, the algorithms listed in Table 3.1 are very inefficient or the global minimum is not found if the estimates are too low[9].

Table 3.1: Number of Evaluations Required to Minimize Simple Example

Žil 1	Žil 2	Strong	Pijav	Brent	Batish
125	165	150	3817	161	816

The LA is able to accurately determine all three global minima with as few as 100 points (i.e., function evaluations) as shown in the figures, whereas the best deterministic algorithm (with significant extra information) took 125 function evaluations. The goal of the results presented in Table 3.1 are to find all three equivalent global minima to within a given x position tolerance of $\epsilon = 0.015$.

3.4 Comparison to Random Search

A numerical example is provided showing faster convergence than random sampling for a simple function. While the chosen function is simple, it should be noted that most algorithms mentioned in the overview provided in Chapter II have no mathematical proof of convergence for any function due to the probabilistic nature and the "black box" structure of the algorithms.

The major difference between the proposed Learning Algorithm and a purely random search is that the Learning Algorithm weights the point selection such that the extrema found so far are more thoroughly explored thanks to the rejection sampling base and successive approximation of the cost function in the LA. The random search only guarantees that a certain number of points in the space are evaluated and requires a very large number of points for a sufficient probability of obtaining the correct optimal value.

The cost function, $J(x) = |x|$, is evaluated for 25 points using the pure random search and the LA algorithm. Figure 3.7 shows the results of each method applied to the absolute value and the histogram of the points. The LA chooses numerous points near the minimum whereas the purely random selection does not. Therefore, the point selection method (rejection sampling) used in the Learning Algorithm, which weights the selection of points toward the minimum value is superior.

A Monte Carlo simulation of 1,000 tests is performed allowing 50 points each to

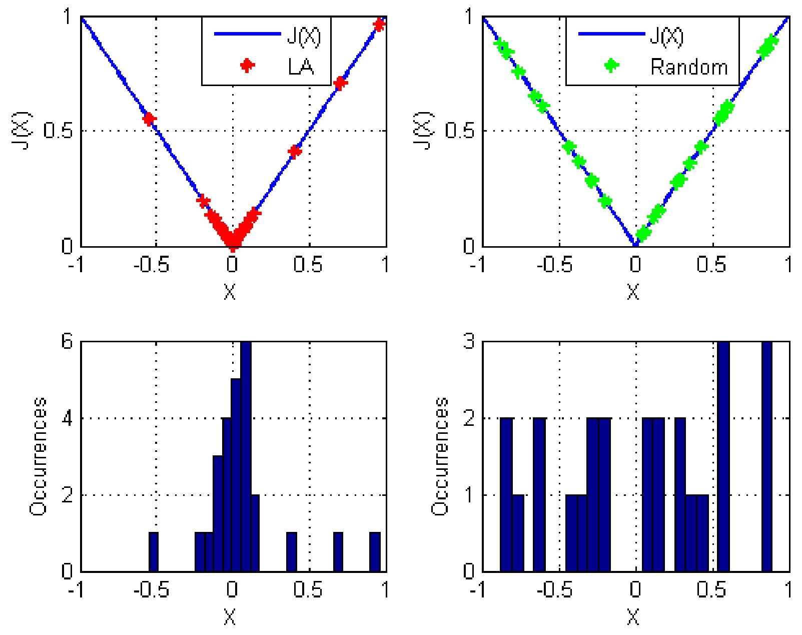

Figure 3.7: Comparison of LA and Random Search for $|x|$ with 25 Points

the Pure Random Search and the Learning Approach. The statistics of the results are showing in Table 3.2. The LA provides superior results in that the mean solution is significantly closer to the true value and the small standard deviation shows a tight grouping around the minimum.

Table 3.2: Statistics of 1,000 Trials on $|x|$

Statistic	Random Search	Learning Approach
Mean $\min\{J(x)\}$	0.0191	$4.6856 \cdot 10^{-6}$
Standard Deviation of $J(x)$	0.0202	0.0006
Best $J(x)$	$2.0128 \cdot 10^{-5}$	$1.0852 \cdot 10^{-16}$
Worst $J(x)$	0.1718	0.0026

A different metric is to determine how many function evaluations (i.e., points) are required to guarantee the solution lies in some small region, $-\epsilon \leq x \leq \epsilon$, near the known minimum, where ϵ is an arbitrary small number.

Lemma 3.4.1. *The expected number of iterations of the random selection algorithm is* $1/\epsilon$*.*

Proof. All our choices are independent from each other and the probability of selecting a point in the interval $(-\epsilon, \epsilon)$ is ϵ. Therefore, the probability of stopping after exactly n iterations is

$$P_n = \epsilon\,(1-\epsilon)^{n-1} \tag{3.3}$$

The expected number of iterations is given by the summation

$$\sum_{n=1}^{\infty} n\epsilon(1-\epsilon)^{n-1} = -\epsilon\left(\sum_{n=1}^{\infty}(1-\epsilon)^{n}\right)' = -\epsilon\left(\frac{1-\epsilon}{\epsilon}\right)' = \frac{1}{\epsilon} \tag{3.4}$$

□

To show the LA converges faster than the pure random search for the absolute value function, another series of Monte Carlo tests are performed and results (average of 10,000 Monte Carlo trials) are shown in Table 3.3. The results show roughly a 30% decrease for larger ϵ, and a significant decrease in the number of function calls for smaller ϵ. This means that as the tolerance, ϵ, is tightened, the LA performs better than the random search, and in general, outperforms the pure random search.

The same numerical test is performed on the function $y = x^2$ as another example. For this case, the distance from ϵ is computed in the independent variable, x. In other words, $error = \min\{|x|\}$ and the goal is to have obtain a point such that $-\epsilon \leq \min\{|x|\} \leq \epsilon$. (Note for the absolute value example above, the distance from

Table 3.3: Numerical Tests for LA and Random Search Applied to $|x|$

ϵ	Random Search (Expected)	Random Search (Numerical)	Learning Approach
0.1	10	10	7
0.01	100	96	43
0.001	1,000	1012	194

zero in the x and y variables is the same.) The results of numerical tests (average of 10,000 Monte Carlo trials) on the function show roughly a 35% decrease for larger ϵ, and roughly a 64% decrease in the number of function calls for smaller ϵ as shown in Table 3.4. These results show numerically that the LA performs better than the random search for two simple functions.

Table 3.4: Numerical Tests for LA and Random Search Applied to x^2

ϵ	Random Search (Expected)	Random Search (Numerical)	Learning Approach
0.1	10	10	6
0.01	100	106	67
0.001	1,000	1005	360

Previously, the LA theory included using a linear interpolation between the nearest points instead of the nearest neighbor. This caused the implemented code to run much slower (as many more computations are necessary) and did not seem to provide an improved increase in performance in terms of accuracy. A comparison of the random search and LA to obtain the minimum of $J(x) = |x|$ with accuracy $\epsilon > 0$

was developed analytically[56]. The proof using the nearest neighbor in the LA, as shown in Algorithm 10 remains a topic for future research.

3.5 Benchmark Problems

It is difficult to define a widely useful *benchmark* problem that does not implicitly favor one algorithm. There is no agreement within the literature on the definition of a benchmark. Thus, it is useful to consider a family of benchmark functions to appreciate the relative merit of competing approaches.

The LA algorithm is used to solve several published, low-dimensional functions. The goal of these applications is to demonstrate that the LA methodology works and make initial performance comparisons with other methods.

It is also difficult to find agreement on *how* to compare the results of different algorithms. One standard method of making the comparisons as fair as possible is to simply compare the number of function evaluations required by the method as the cost function is typically the most computationally costly portion of the optimization problem. To this end, the comparisons given in this dissertation consist of the number of cost function evaluations and/or the accuracy achieved after a given, fixed number of function evaluations. These results demonstrate that the LA's costs are equivalent to the number of points accepted.

Comparisons are made (given a fixed number of cost function evaluations) with the widely used genetic algorithm (GA) on each 2-D problem. The GA used for these problems is in MATLAB (R2007a) [57] with the default parameter set–the population size and number of iterations are specified per problem. It should be emphasized that the GA in MATLAB is highly optimized, built-in code making the timing comparisons difficult. For more general information on the GA, the reader may refer to Reference [29].

Further information on benchmarking the performance of competing algorithms can be found in Refs. [3, 12, 58, 59, 60]. Most of the problems shown are 2-D with extensions to higher dimensions, with some examples in higher dimensions, but the 2-D problems illustrate well the power of the LA as the results can be shown graphically and the reader can visually see the results. The purpose of this section is to test the LA on a number of functions that are considered benchmark problems with various characteristics in order to validate the LA against the known functions and other optimization algorithms. Higher dimensional problems (some with no known solutions) are examined in the next chapter.

3.5.1 Spherical Cost Function

A very simple test problem is the sphere model[7], also known as DeJong's First Function. No matter the dimensional size, this function always has a single minimum located at the origin, $\mathbf{x} = \mathbf{0}$, with an associated cost value of $J(\mathbf{x}) = 0$. The simplicity of this function allows it to be tested in any dimension. The function is written as

$$J(\mathbf{x}) = \sum_{i=1}^{N} x_i^2 \tag{3.5}$$

with variable bounds $-5 \leq x_i \leq 5$, where in 2-dimensions, $x_1 = x$ and $x_2 = y$. Although this function is separable, this is not taken advantage of in the LA. In addition, the spherical cost function is tested in higher dimensions below.

Figure 3.8 shows the function while Fig. 3.9 shows the contour plot with accepted points from the LA algorithm (note the global minima are in black).

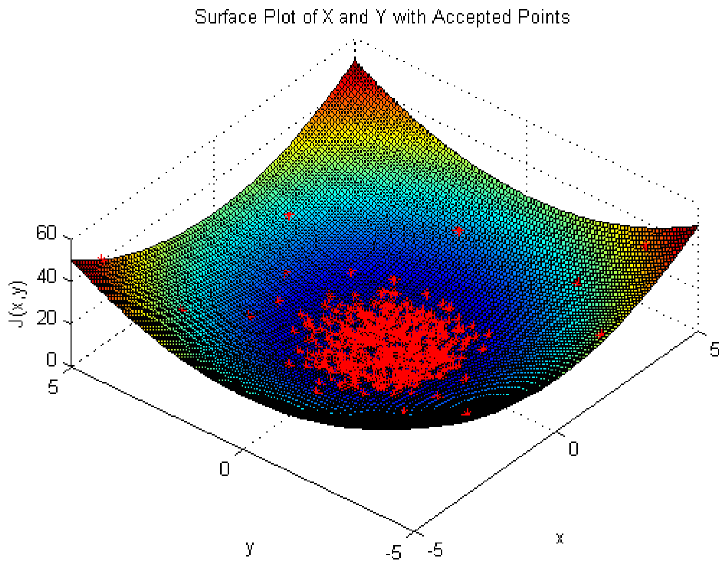

Figure 3.8: Sphere Function 3-D Plot

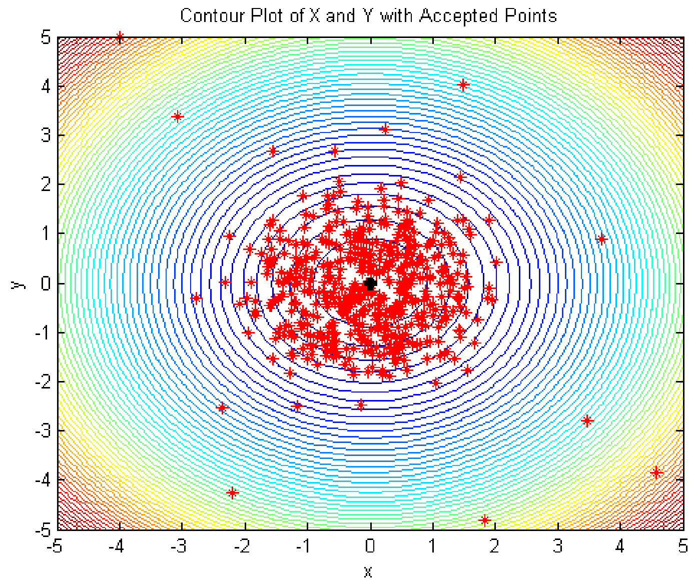

Figure 3.9: Sphere Function Contour Plot

Table 3.5 shows the results of a Monte Carlo analysis where 1,000 tests are run. The LA is compared to a Genetic Algorithm with different population and number of generations. As the GA requires parameters to be tuned, two different parameter sets are used. In the table, the population size is the first number in parenthesis followed by the number of generation (e.g., GA (population/generations)). The LA is allowed 500 function calls (i.e., points), as the GA is allowed as many (population multiplied by generations is equal to the number of function calls). The mean, standard deviation, best, and worst values are shown from the Monte Carlo results.

Table 3.5: Results of Monte Carlo Analysis of Sphere Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	0.0016	$1.1285 \cdot 10^{-4}$	$1.0357 \cdot 10^{-7}$
1σ of $J(\mathbf{x})$	0.0031	$4.0673 \cdot 10^{-4}$	$1.1801 \cdot 10^{-6}$
Best $J(\mathbf{x})$	$4.3747 \cdot 10^{-8}$	$2.9393 \cdot 10^{-7}$	$3.1430 \cdot 10^{-11}$
Worst $J(\mathbf{x})$	0.0016	0.0033	$4.3012 \cdot 10^{-4}$
Mean CPU Time (sec)	0.1220	0.1262	0.1256
Max CPU Time (sec)	0.4863	0.4524	0.2028

The CPU time is computed using the MATLAB built in timing function `cputime` which returns the elapsed time used by the CPU during the optimization process. While this method may be neither exact nor optimal, it does provide an initial comparison. The computer specifications included a Pentium Dual-Core T4300 at 2.10 GHz and 4 GB or RAM. The GA used in the comparison is the built in MATLAB GA, which is highly optimized and efficient. Thus, the fact that the LA competes in mean speed and is 50% better in maximum speed is an indication of the efficiency of the LA algorithm. As this section is meant to demonstrate the LA theory applied some benchmark functions, the timing is not considered in all cases. In the applications in Chapter 4, the timing is considered where appropriate.

3.5.2 Goldstein and Price Function

The Goldstein and Price Function, as adopted from Dixon and Szegö [7], is written as

$$\begin{aligned} J(\mathbf{x}) &= [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)] \times \\ &\quad [30 + (2x - 3y)^2(18 - 32x + 12x^2 - 48y - 36xy + 27y^2)] \qquad (3.6) \end{aligned}$$

This equation is difficult for many optimizers because there is a large hill in a portion of the search space, but most of the search space is a large, flat plateau around the global minimum.

The bounds used are $-2 \leq x, y \leq 2$. The known global minimum is at $(x, y) = (0, -1)$ with $J(x, y) = 3$. The LA found a value of $(x, y) = (-0.0440, -0.9762)$ which gave $J(x, y) = 3.8951$ on a single run of 200 accepted points (shown). Figure 3.10 shows the function with the accepted points and Figure 3.11 shows the contour plot with the accepted points. This function has a broad, weak minimum and is useful to discriminate effectiveness of sample (or other) optimization algorithm's ability to accurately isolate weak extrema.

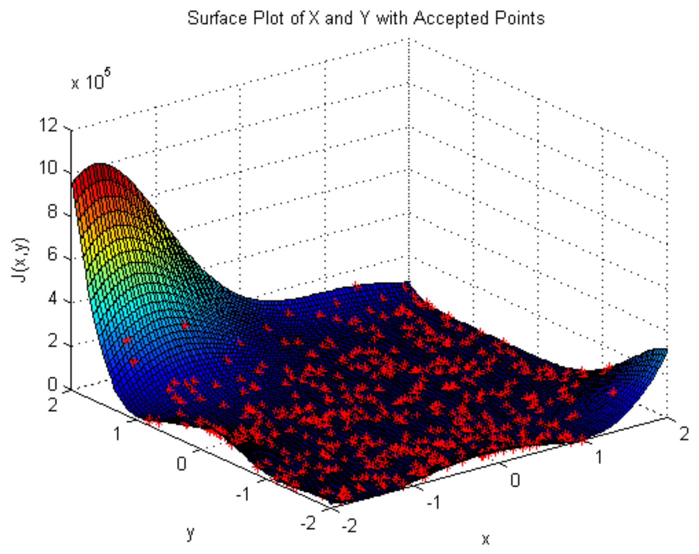

Figure 3.10: Goldstein and Price Function 3-D Plot

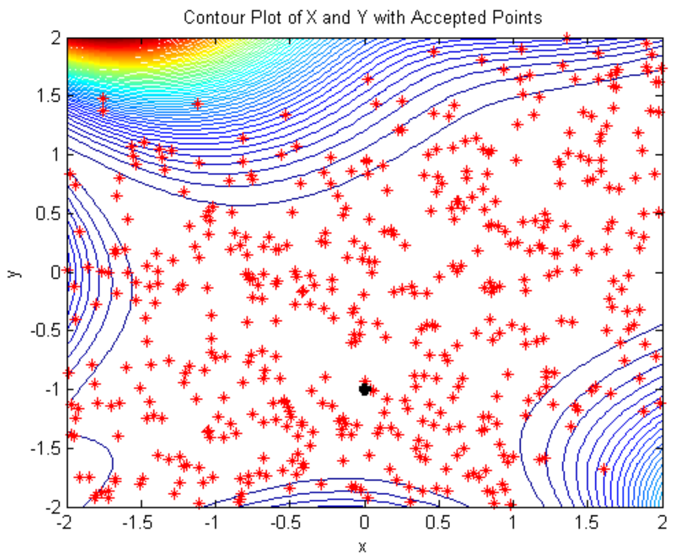

Figure 3.11: Goldstein and Price Function Contour Plot

Table 3.6 shows the results of a Monte Carlo analysis where 1,000 tests are run. Statistics of the Monte Carlo optimization are shown with the same parameter meaning as in Table 3.5.

Table 3.6: Results of Monte Carlo Analysis of Goldstein and Price Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	7.1553	17.9021	3.0687
1σ of $J(\mathbf{x})$	7.8116	30.4037	0.3472
Best $J(\mathbf{x})$	3.0011	3.0010	3.0000
Worst $J(\mathbf{x})$	38.2222	93.9786	4.2602
Mean CPU Time (sec)	0.1236	0.1273	0.1238
Max CPU Time (sec)	0.4306	0.4457	0.1716

3.5.3 Rosenbrock's Function

Rosenbrock's function (also known as Rosenbrock's banana function) is a well-known and widely-used test function for optimization techniques[61, 7]. The global minimum lies in a parabolic shaped valley at $(x, y) = (1, 1)$. Many optimization methods (both deterministic and stochastic) have success in identifying the valley as the region of the global optimum, but have great difficulty in finding a point near the global optimum due to the sharp *horseshoe canyon*, nonlinear behavior. The function is written

$$J(\mathbf{x}) = (1 - x)^2 + 100(y - x^2)^2 \tag{3.7}$$

with the bounds $-2 \le x \le 2$ and $-1 \le y \le 3$.

The LA was applied to Rosenbrock's function and in one particular run of the algorithm which accepted 200 points, the (geometrically) closest point to the known minimum is $(x, y) = (1.0314, 0.9991)$ with a cost function value of $J(x, y) = 0.4211$. However, the point found with the lowest cost function was $(x, y) = (0.8496, 0.7130)$ with a cost function value of $J(x, y) = 0.0305$. Figure 3.12 shows Rosenbrock's function and the accepted points from the LA while Figure 3.13 shows a contour plot of the function and the accepted points (the black point at $(x, y) = (1, 1)$ is the known global minimum).

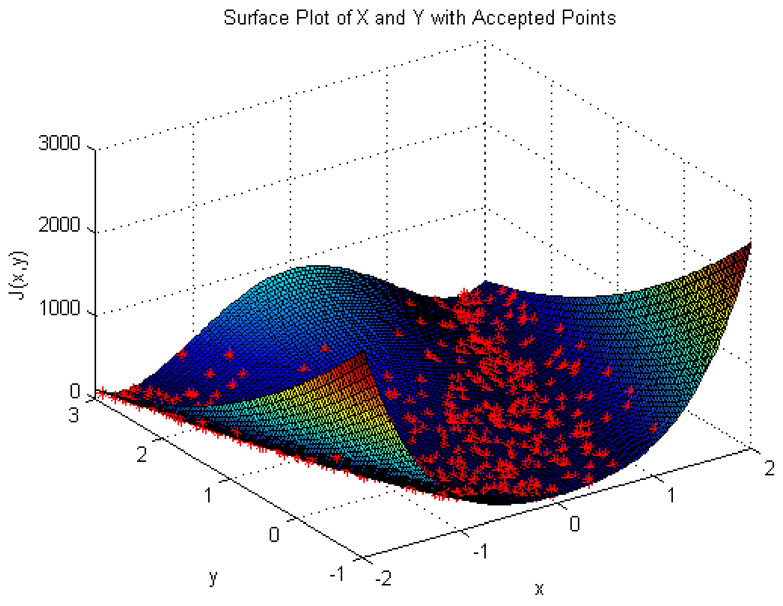

Figure 3.12: Rosenbrock Function 3-D Plot

A Monte Carlo simulation, where 1,000 tests are conducted in which the stopping criteria is 500 accepted points for the LA (or 25 generations of 20 points each for the GA) was performed. Figure 3.14 shows a histogram of the minimum cost function value for the 1,000 trials comparing the results of the LA with the GA.

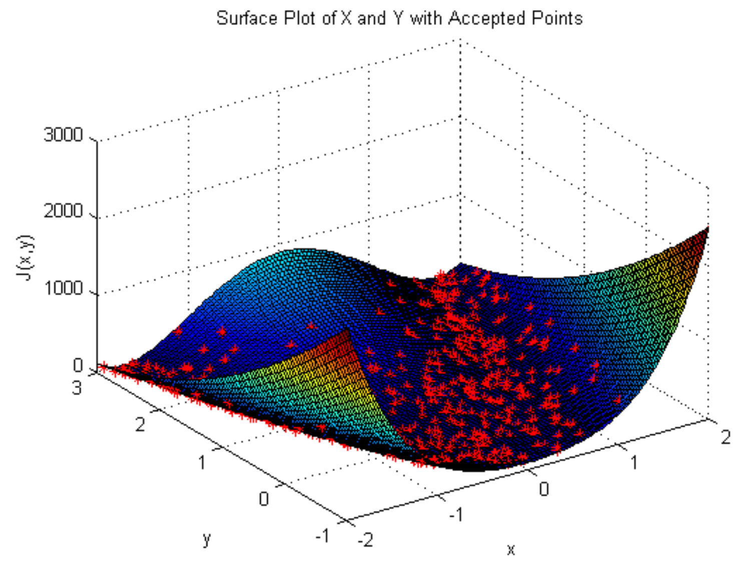

Figure 3.13: Rosenbrock Function Contour Plot

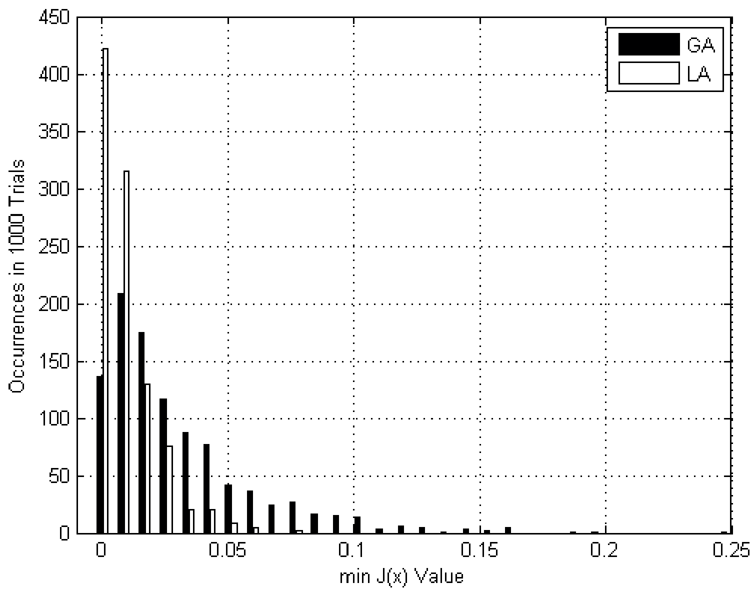

Figure 3.14: Rosenbrock Function Monte Carlo Comparison of GA and LA

Statistics comparing the LA and GA in a Monte Carlo analysis where 1,000 tests are run are shown in Table 3.7. The meaning of the parameters is the same as in Table 3.5.

Table 3.7: Results of Monte Carlo Analysis of Rosenbrock's Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	0.0285	0.0462	0.0048
1σ of $J(\mathbf{x})$	0.0379	0.0548	0.0035
Best $J(\mathbf{x})$	$6.4974 \cdot 10^{-5}$	$1.9567 \cdot 10^{-6}$	$6.0010 \cdot 10^{-6}$
Worst $J(\mathbf{x})$	0.2186	0.2542	0.0900
Mean CPU Time (sec)	0.1530	0.1426	0.1423
Max CPU Time (sec)	0.4386	0.4140	0.1716

3.5.4 Rastrigin Function

The Rastrigin Function, first published by Rastrigin in 1974 (in Russian) and here as adopted from Reference [9], is written as

$$J(\mathbf{x}) = x^2 + y^2 - \cos(18x) - \cos(18y) \tag{3.8}$$

in the range $-1 \leq x, y \leq 1$. The known global minimum is at $(x, y) = (0, 0)$ with $J(x, y) = -2$.

There are fifty local minima arranged in a lattice configuration in the search space, which makes this function difficult for many algorithms which quickly get caught in a local, non-optimal minimum. Figures 3.15 and 3.16 show the results when the LA was used with 200 accepted points. Note that the global optimum is well searched as are several of the near-global values. This function is difficult to find all local minimum values with only 200 points, but the LA performs well.

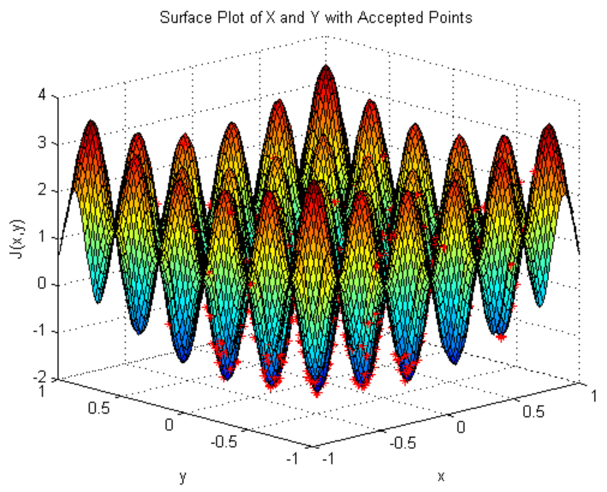

Figure 3.15: Rastrigin Function 3-D Plot

Table 3.8 shows the results of a Monte Carlo analysis where 1,000 tests are run and compares the LA and GA with two different parameter sets.

Table 3.8: Results of Monte Carlo Analysis of Rastrigin Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	-1.8979	-1.8660	-1.9975
1σ of $J(\mathbf{x})$	0.0857	0.1149	0.0156
Best $J(\mathbf{x})$	-2.0000	-2.0000	-2.0000
Worst $J(\mathbf{x})$	-1.6447	-1.3942	-1.9840
Mean CPU Time (sec)	0.1236	0.1329	0.1466
Max CPU Time (sec)	0.4221	0.4297	0.2496

Reference [9] provided results of minimizing the Rastrigin function with the fol-

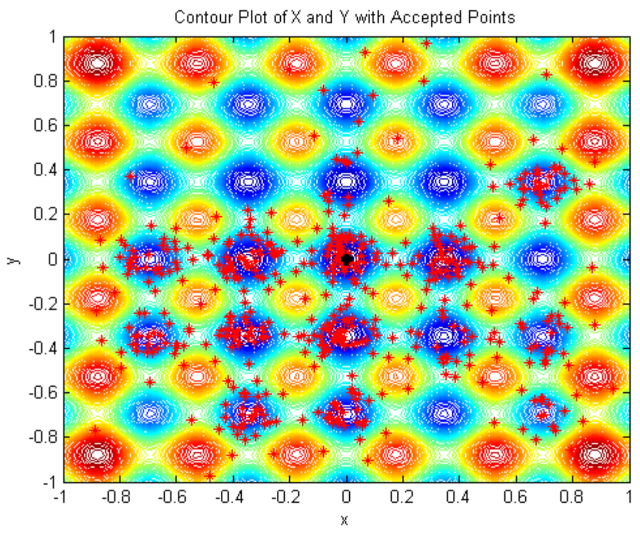

Figure 3.16: Rastrigin Function Contour Plot

lowing algorithms with a stopping accuracy of $x^2 + y^2 \leq 2.1518 \cdot 10^{-4}$:

1. Pure Random Search (PRS)
2. Multistart Random Search (MS)
3. Competing points with random local search (CRS)
4. Multistart with gradient local search (MSG)
5. Competing points with gradient local search (CG)
6. Strongin's approach (1972) based on a statistical local algorithm with a model reduction[9].[1] (ST)
7. P^*-algorithm of Žilinskas, a statistical based method[9] (Žil 1)

[1]The search space and required accuracy were reduced

Table 3.9, below, is partially reproduced from and adapted from Table 8.10 and other results presented in Section 8.2.3 of Ref. [9] and shows the average number of function evaluations required to minimize the Rastrigin function. The LA is run 1,000 times and the average number of points (i.e., function calls) is reported as well. The Žil 1 algorithm (P^*-algorithm of Žilinskas) was stopped after 150 iterations

Table 3.9: Average Number of Function Evaluations to Minimize Rastrigin Function

PRS	MS	CRS	MSG	CG	ST	Žil 1	LA
5917	1176	184	556	111	240	179	89

and refined by a local search which required an average of 29 iterations to reach a tolerance within 0.0015 of the known global minimum.

3.5.5 Modified Ackley Function

The Modified Ackley Function[3, 62], as generalized by Bäck[6] is a widely used benchmark function for numerical optimization as it can easily be extended into a large number of dimensions (which is shown at the end of this section) and it has multiple minima. The equation is written as

$$J(\mathbf{x}) = -c_1 \exp\left(-c_2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(c_3\, x_i)\right) + c_1 + \mathrm{e} \tag{3.9}$$

For now, only $N = 2$ will be considered so that the reader can visualize the results through the figures. Common values of the constants are given as in Ref. [3] and are used for this simulation

$$c_1 = 20 \quad c_2 = 0.2 \quad c_3 = \pi.$$

Note that the global minimum occurs at $(x, y) = (0, 0)$.

The LA applied to the Modified Ackley Function is shown in Figs. 3.17 and 3.18 over the range $-4 \leq x, y \leq 4$. In this particular run of the algorithm, the minimum cost function value found is $(x, y) = (-0.0053, 0.0156)$ with $J(x, y) = 0.0484$.

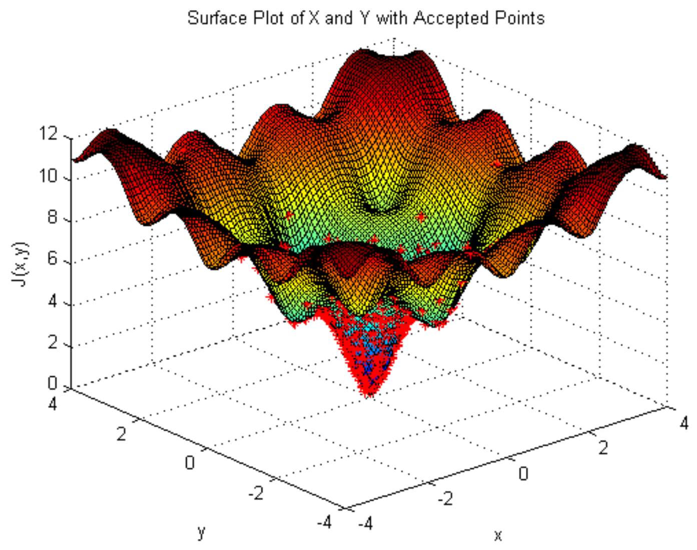

Figure 3.17: Modified Ackley Function 3-D Plot

The literature disagrees on what the search space bounds, Ω, should be for the Ackley function. As such, two different sets are studied. The reason for increasing the bounds is to add a significant number of local minima. When the bounds are increased to $10 \leq x, y \leq 10$, the LA found a value of $J(x, y) = 0.0619$. Figures 3.19 and 3.20 show those results.

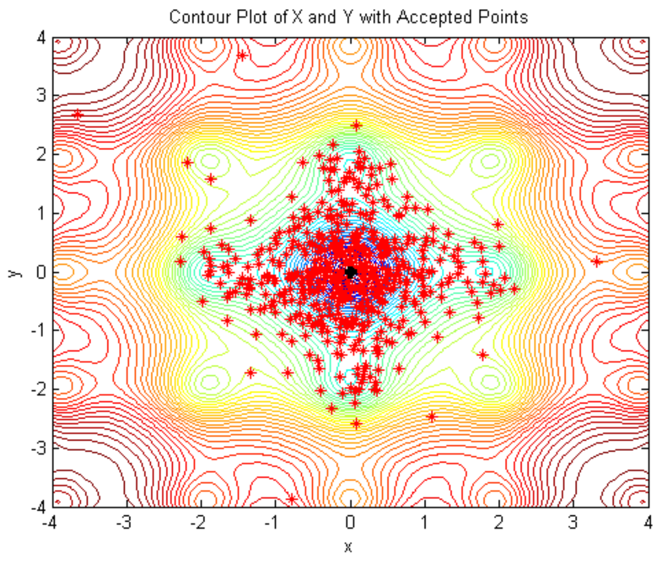

Figure 3.18: Modified Ackley Function Contour Plot

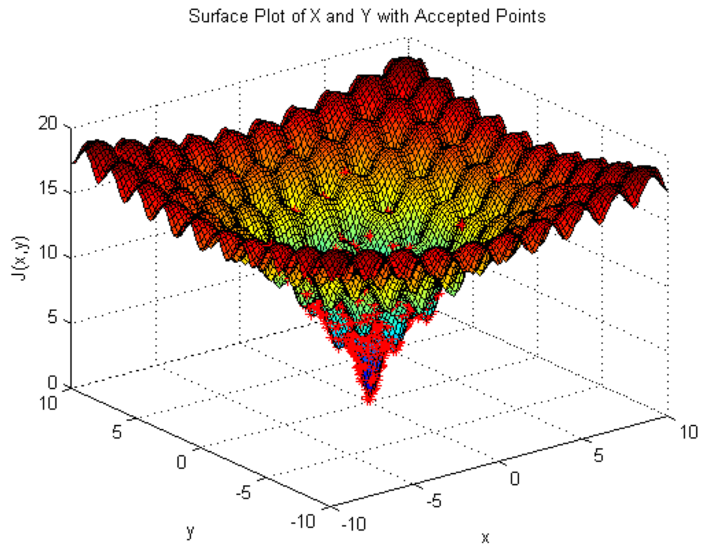

Figure 3.19: Modified Ackley Function 3-D Plot, Extended Bounds

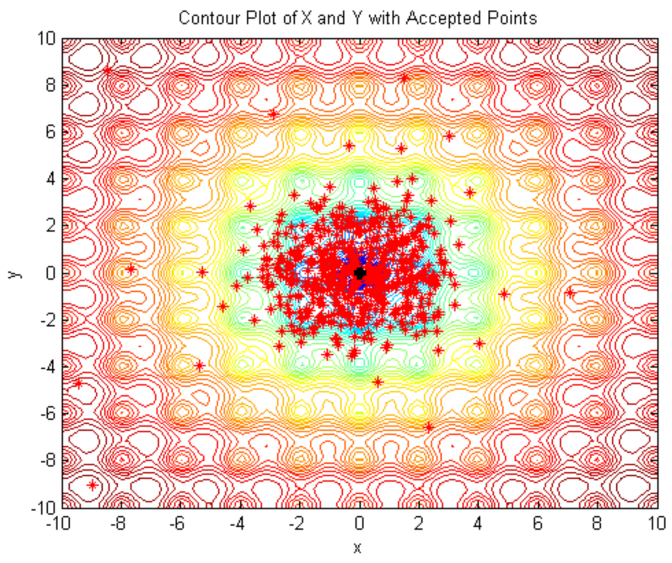

Figure 3.20: Modified Ackley Function Contour Plot, Extended Bounds

Again, a Monte Carlo simulation is run with 1,000 trials with 500 allowed function calls. The minimum cost function value achieved by the LA and GA (with population 25 and 20 generations) are compared in the histogram shown in Fig. 3.21.

Table 3.10 shows the statistical results of the Monte Carlo analysis. The data in the table represents Ackley's function investigated over the search space $-4 \leq x, y \leq 4$.

Table 3.10: Results of Monte Carlo Analysis of Modified Ackley Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	0.0756	0.0177	0.0076
1σ of $J(\mathbf{x})$	0.0509	0.0219	0.0031
Best $J(\mathbf{x})$	0.0027	$1.9684 \cdot 10^{-5}$	$2.7037 \cdot 10^{-8}$
Worst $J(\mathbf{x})$	0.2699	0.1576	0.1309

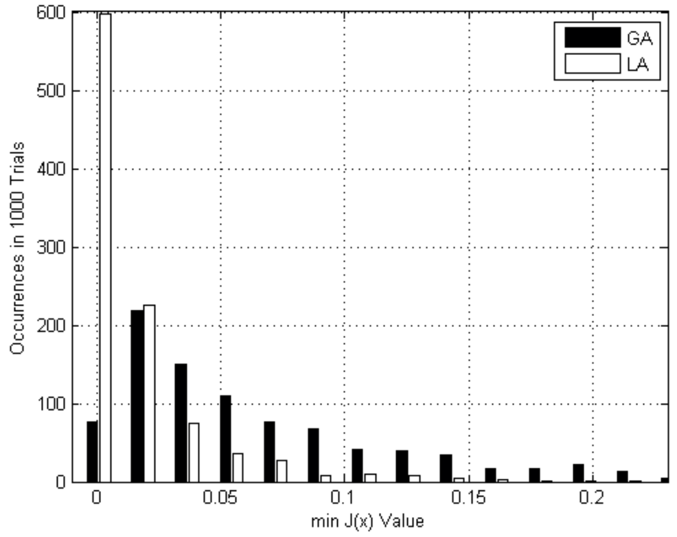

Figure 3.21: Modified Ackley Function Monte Carlo Comparison of LA and GA

3.5.6 Adjiman Function

One difficulty faced by many stochastic algorithms is when the global minimum is on an edge of the search space. The function attributed to Adjiman is a simple sine and cosine function[7, 63] which is written as

$$J(\mathbf{x}) = \cos(x)\sin(y) - \frac{x}{y^2+1} \tag{3.10}$$

with variable bounds $-5 \leq x, y \leq 5$. The global minimum value occurs at $(x, y) = (5, 0)$ with a cost function value of $J(x, y) = -5$. The interesting fact about this function is that independent from the range selected, the global minimum always occurs at the upper bound of x and at $y = 0$, while the global minimum value is $J(x, y) = -[\max(x)]$. Thus the scale of the problem is changed without losing information on the global minimum position and value.

Figure 3.22 shows the function while Figure 3.23 shows the contour plot with accepted points from the LA algorithm (note the global minimum is in black). It can be seen from the figures that the LA collapses quickly to the global minimum in this case. Table 3.11 shows the results of a Monte Carlo analysis where 1,000 tests

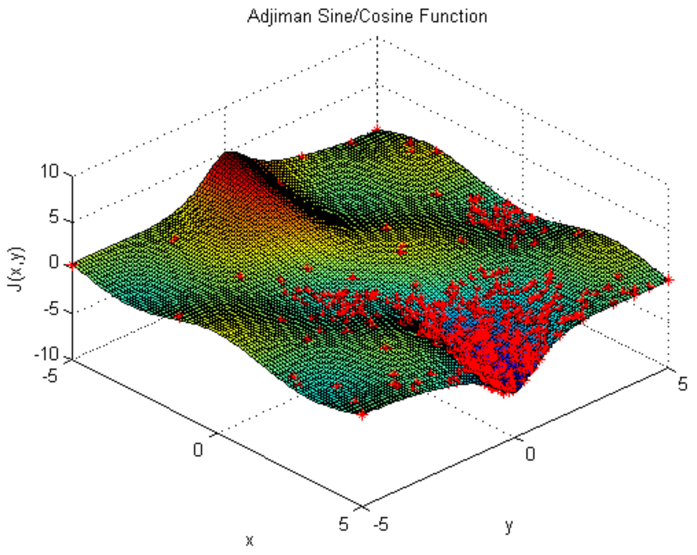

Figure 3.22: Adjiman Function 3-D Plot

are run. The mean, standard deviation, best, and worst values are shown and the labels are the same as in Table 3.5.

Table 3.11: Results of Monte Carlo Analysis of Adjiman Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	-4.4955	-4.8016	-4.9363
1σ of $J(\mathbf{x})$	0.4190	0.2384	0.0347
Best $J(\mathbf{x})$	-4.9938	-5.0000	-5.0000
Worst $J(\mathbf{x})$	-3.1585	-3.7180	-4.8802

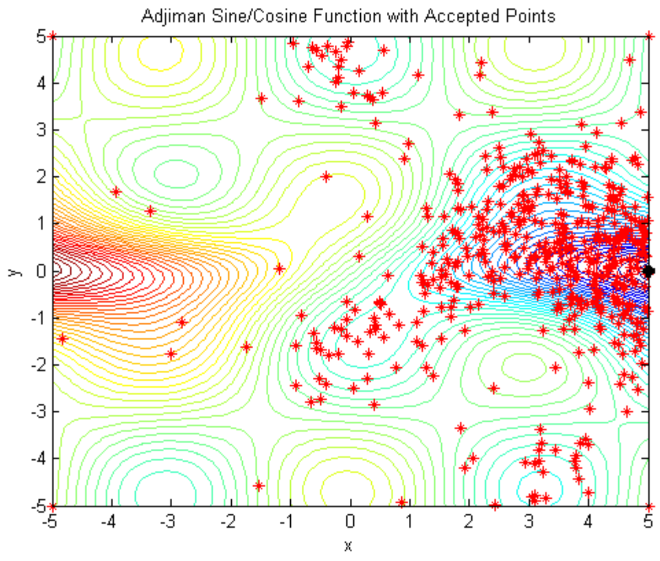

Figure 3.23: Adjiman Function Contour Plot

3.5.7 A Discontinuous Function

Deterministic algorithms frequently fail when the function being optimized is discontinuous. A one-dimensional test function is generated where the global minimum is located at $(x, y) = (6, -1.1591)$. The function is written as

$$J(x) = \begin{cases} \dfrac{x}{3} \sin x \cos x & x \in [0, 6) \\ \dfrac{x}{4} \sin 0.9x & x \in [6, 10] \end{cases} \tag{3.11}$$

The bounds used are $0 \leq x \leq 10$ and $-1.5 \leq y \leq 2.5$. As seen in Figs. 3.24 and 3.25, the LA is able to find the correct minimum with as few as 25 points. The minimum point found after only 25 iterations is $(x, y) = (6.0001, -1.1591)$. In addition to identifying the correct location of the global minimum, the local minimum near $x \approx 2.5$ and $x \approx 5.5$ are identified, as can be seen in the figures.

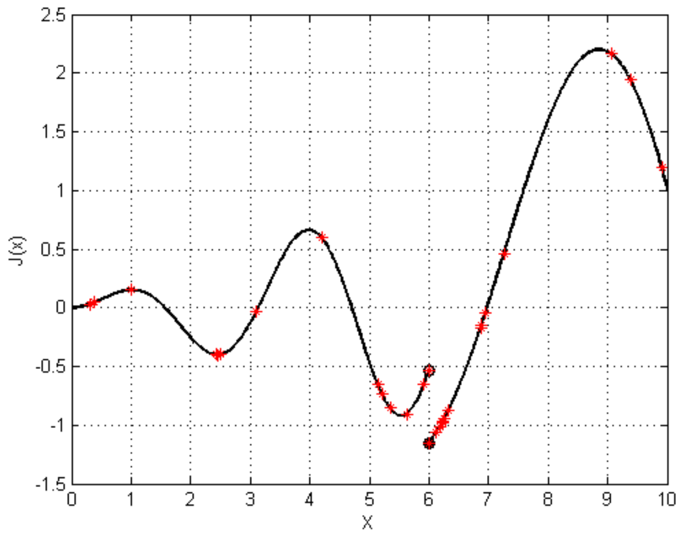

Figure 3.24: Discontinuous Function, 25 Accepted Points

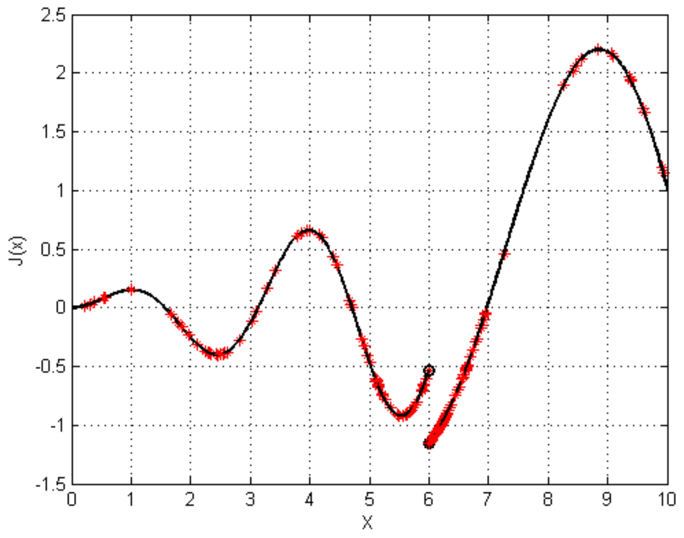

Figure 3.25: Discontinuous Function, 100 Accepted Points

A standard GA is run on the discontinuous function shown in Eq. 3.11. Figure 3.26 shows the final population (of size 100) after 10 generations (i.e., 1,000 function calls). A Monte Carlo test is run for 1,000 trials of the GA with the population of 100 and 10 generations. Figure 3.27 shows a histogram of the x position that produced the minimum value of $J(x)$. It should be noted that 980 of the 1,000 Monte Carlo trials ended with the GA finding the non-global optimal location of $x \approx 2.5$, six trials ended at the non-global location of $x \approx 5.5$, and only 13 of the trials found the correct global minimum of $x = 6$ (and one value ended at $x \approx 5$). Comparatively, the LA Monte Carlo analysis with 1,000 trials ended with all 1,000 trials identifying the correct global minimum at $x = 6$, to four significant decimal places, using only 100 points–an order of magnitude less function calls than the GA.

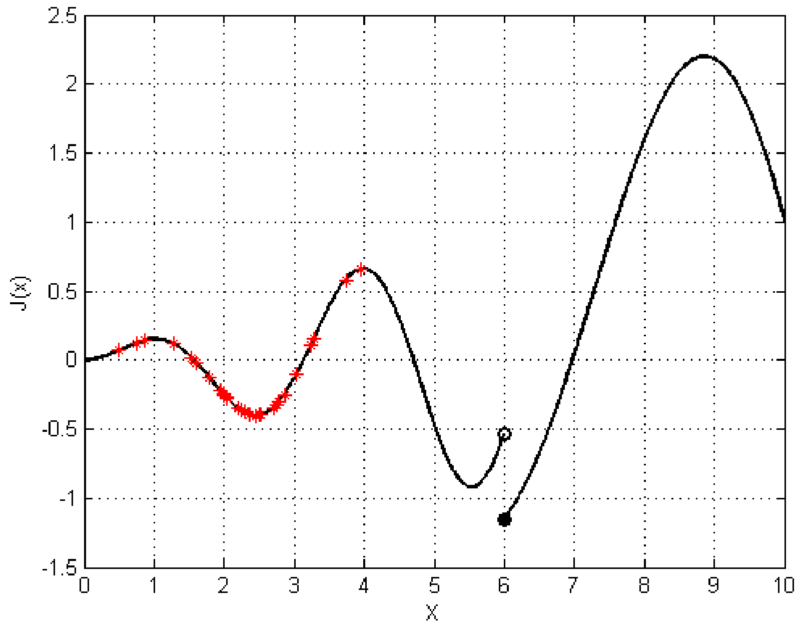

Figure 3.26: Discontinuous Function, Final GA Population

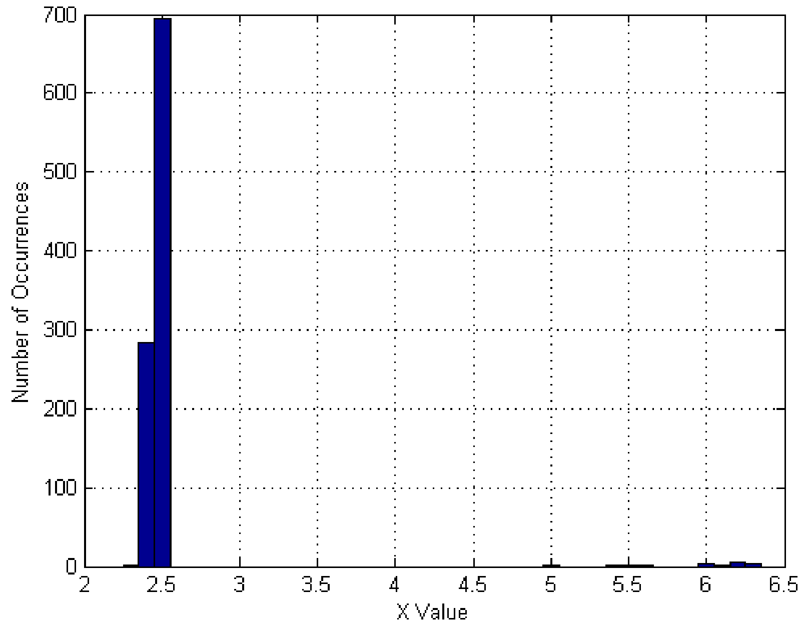

Figure 3.27: Discontinuous Function, GA Monte Carlo Results

3.5.8 Camel Back Functions

The Camel Back Functions (with three and six Humps) are lesser known benchmarks taken from Ref. [63]. The functions are described below.

3.5.8.1 Three Hump Camel Back Function

The Three Hump Camel Back Function is written as

$$J(\mathbf{x}) = 2x^2 - 1.04x^4 + \frac{x^6}{6} - xy + y^2 \tag{3.12}$$

Common variable bounds are $-2 \leq x, y \leq 2$ and the global minimum value occurs at $(x, y) = (0, 0)$ with a cost function value of $J(x, y) = 0$.

Figure 3.28 shows the function while Fig. 3.29 shows the contour plot with accepted points from the LA algorithm (note the global minimum is in black).

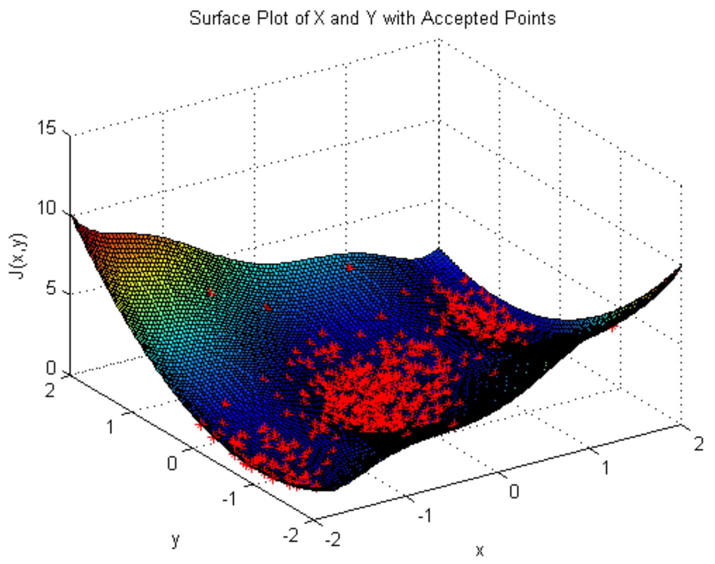

Figure 3.28: Three Hump Camel Back Function 3-D Plot

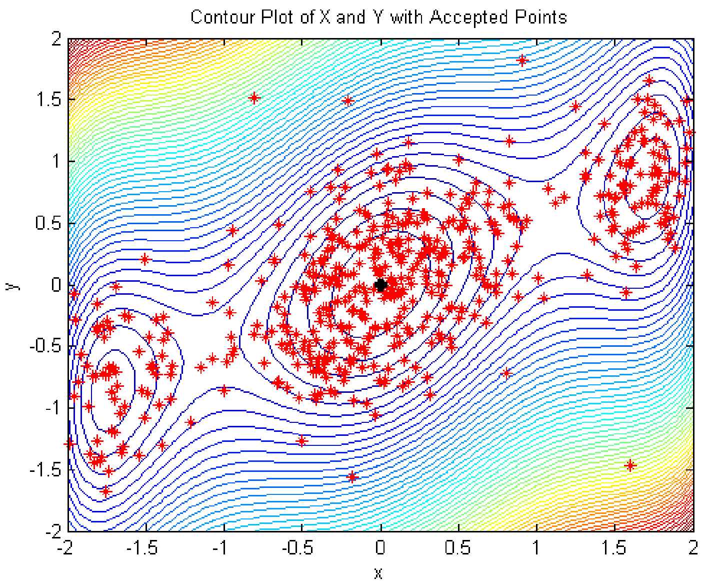

Figure 3.29: Three Hump Camel Back Function Contour Plot

Table 3.12 shows the statistical results of a Monte Carlo analysis where 1,000 tests are run comparing the LA and GA. For this function, the GA with population 25 and 20 generations gave a smaller (i.e., more accurate) mean than the LA. However, this difference is in the fourth decimal place. The LA shows a tighter grouping or solutions in that the standard deviation is smaller and the worst cost function out of the 1,000 trials is lower for the LA than the GA.

Table 3.12: Results of Monte Carlo Analysis of Three Hump Camel Back Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	0.0013	$2.4237 \cdot 10^{-4}$	$3.4587 \cdot 10^{-4}$
1σ of $J(\mathbf{x})$	0.0028	$7.5156 \cdot 10^{-4}$	$4.2381 \cdot 10^{-5}$
Best $J(\mathbf{x})$	$4.8918 \cdot 10^{-6}$	$9.4629 \cdot 10^{-9}$	$1.2101 \cdot 10^{-9}$
Worst $J(\mathbf{x})$	0.0218	0.0057	0.0012

3.5.8.2 Six Hump Camel Back Function

Branian published the Six Hump Camel Back Function in 1972, and it is a bi-modal benchmark function[7, 63] written as

$$J(x, y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right) x^2 + xy + \left(-4 + 4y^2\right) y^2 \tag{3.13}$$

with variable bounds $-1.9 \leq x \leq 1.9$ and $-1.1 \leq y \leq 1.1$. The equivalent global minima values occur at $(x, y) = (-0.0898, 0.7126)$ and $(0.0898, -0.7126)$ with a cost function value of $J(x, y) = -1.0316$. Note that the function is symmetric about the a point, in particular the origin, meaning that

$$J(x, y) = J(-x, -y)$$

Figure 3.30 shows the function while Fig. 3.31 shows the contour plot with accepted points from the LA algorithm (note the global minima are in black).

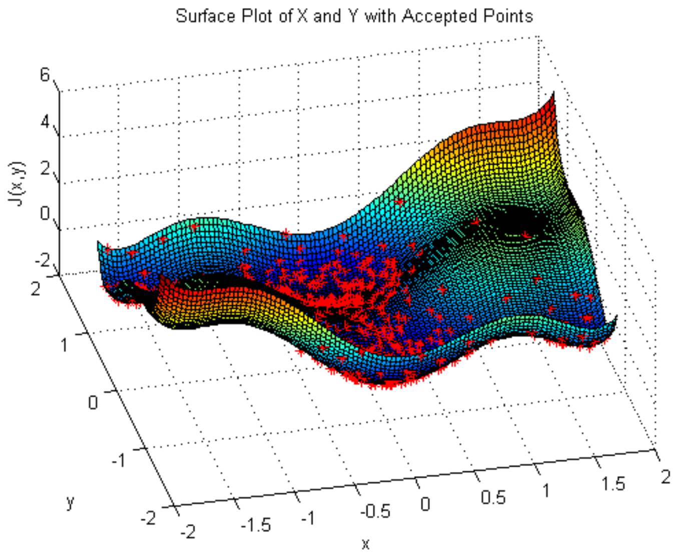

Figure 3.30: Six Hump Camel Back Function 3-D Plot

The difficulty in comparing the LA with the GA for this function is that the GA is strongly polarized by the best point in the population in a given iteration. The standard GA is not well designed for finding multi-minima, as is demonstrated with this example.

In 10,000 trials of the GA as applied to the Six Hump Camel Back Function, 3,124 trials went to the minimum value at $(x, y) = (-0.0898, 0.7126)$ and the remaining 6,876 trials found the minimum value located at $(x, y) = (0.0898, -0.7126)$. This

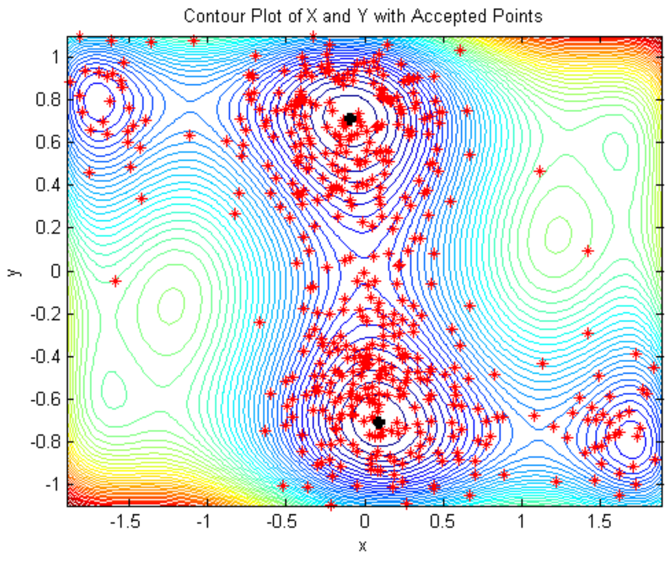

Figure 3.31: Six Hump Camel Back Function Contour Plot

result shows that the standard GA is biased toward the region of the minimum whose vicinity was sampled first.

Table 3.13 shows the statistical results of a Monte Carlo analysis where 1,000 tests are run and the GA and LA are compared. The comparison reported in the table is made by finding the single best cost function value, meaning the point which provides a solution closest to $J(x, y) = -1.0316$.

Table 3.13: Results of Monte Carlo Analysis of Six Hump Camel Back Function

Statistic	GA (50/10)	GA (25/20)	LA
Mean $J(\mathbf{x})$	-1.0253	-1.0312	-1.0315
1σ of $J(\mathbf{x})$	0.0096	0.0018	0.0009
Best $J(\mathbf{x})$	-1.0316	-1.0316	-1.0316
Worst $J(\mathbf{x})$	-0.9718	-1.0154	-1.0297

3.5.9 Higher Dimensional Functions

Two previously tested functions and two new functions are now investigated in higher dimensions. Results of the Sphere Function and Modified Ackley Function in higher dimensions and the three and six dimensional Hartman functions are shown in the remainder of this section. The purpose of this exercise is to show the LA in higher dimensions (up to 6). The examples of the next chapter show the LA algorithm used on significantly higher dimensional problems.

3.5.9.1 Sphere Function in Higher Dimensions

Although the Sphere Function is simple, it allows for any dimension, N, to be investigated quickly. Equation (3.5) has the global minimum value at $\mathbf{x} = \mathbf{0}$ and $J(\mathbf{x}) = 0$ for any dimensional space. Table 3.14 shows the results of 1,000 runs of the LA algorithm in higher dimensions.

Table 3.14: Minima of Sphere Function in Higher Dimensions

Dimension	Accepted Points	mean $J(\mathbf{x})$	1-σ
3	500	0.0150	
3	1,000	0.0106	
3	3000	0.0013	
4	1,000	0.0226	
4	4000	0.0084	
5	5000	0.0321	

3.5.9.2 Ackley's Function in Higher Dimensions

Similar to the Sphere Function, the Modified Ackley's function as written in Eqn. (3.9) is written in any dimension. For any dimensional size, the global minimum is located at $\mathbf{x} = \mathbf{0}$ and $J(\mathbf{x}) = 0$. Table 3.15 shows the results of 1,000 runs of the LA algorithm in higher dimensions.

Table 3.15: Minima of Modified Ackley Function in Higher Dimensions

Dimension	Accepted Points	mean $J(\mathbf{x})$	1-σ
3	500	0.9845	0.0185
3	1,000	0.5471	0.0054
3	3,000	0.1032	0.0011
4	1,000	0.7076	0.0124
4	4,000	0.0846	0.0012
5	5,000	0.2114	0.0237

3.5.9.3 Hartman 3-D

The three dimensional function, as taken from Dixon and Szegö [7], has as the global minimum $\mathbf{x} = (0.114614, 0.555649,\ 0.852547)$ and $J(\mathbf{x}) = -3.86278$. The function is written as

$$J(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left[-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right] \tag{3.14}$$

with the following values

$$\alpha = [1, 1.2, 3, 3.2]^{\mathrm{T}} \qquad A = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix} \qquad P = 10^{-4} \cdot \begin{bmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381.5 & 5743 & 8828 \end{bmatrix}$$

in the search space $0 \leq x_i \leq 1$ for $i = 1, 2, 3$. Over the search space, there are four local minima, and the single global minimum listed above. The table below shows the number of function evaluations (i.e., points) required to minimize the 3-dimensional Hartman function as shown by Boender[8]. The LA is tested 1,000 times and the average number of function calls to find the global minimum to 6 significant figures is shown.

Table 3.16: Number of Function Evaluations for Hartman 3-D

Method	Function Calls
Törn	2584
De Biase	732
Price	2400
LA	461

3.5.9.4 Hartman 6-D

The six dimensional Hartman function, as taken from Dixon and Szegö [7], has as the global minimum $\mathbf{x} = (0.201690, 0.150011,\ 0.476874, 0.275332, 0.311652, 0.6573)$ and $J(\mathbf{x}) = -3.32237$. The function is written as

$$J(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left[-\sum_{j=1}^{6} B_{ij}(x_j - Q_{ij})^2\right] \tag{3.15}$$

with the following values

$$\alpha = [1, 1.2, 3, 3.2]^{\mathrm{T}} \qquad B = \begin{bmatrix} 10 & 3 & 17 & 3.05 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$$

$$Q = 10^{-4} \cdot \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$$

in the search space $0 \leq x_i \leq 1$ for $i = 1, 2, 3$. Over the search space, there are six local minima. The table below shows the number of function evaluations (i.e., points) required to minimize the 6-dimensional Hartman function as shown by Boender[8]. The LA was tested 1,000 times and the average number of function calls to find the global minimum to 6 significant figures is shown.

Table 3.17: Number of Function Evaluations for Hartman 6-D

Method	Function Calls
Törn	3447
De Biase	806
Price	7600
LA	710

As can be seen from Tables 3.16 and 3.17, the LA requires fewer function calls than methods previously tested in the literature.

3.5.10 Comments on Benchmark Functions

In this section, examples of popular benchmark functions taken from multiple sources in the literature are shown. The LA algorithm is tested against eight 2-D functions, a discontinuous 1-D function, and four higher dimensional functions. In all cases, the LA is compared with the GA, and in some cases, results of other methods were taken from the literature.

From the analysis of the testing, the LA performed better in terms of accuracy (given a fixed number of points) and generally performed comparable or better in computational speed. When compared to other methods, the LA is able to find a minimum value (to within tolerance) in significantly fewer cost function evaluations.

3.6 Modifications for Local Search

One of the unstated goals of this research is to develop an optimization technique that did not require tuning of parameters, which given different values in EAs can greatly effect the result. For example, many experienced practitioners of genetic algorithms run a Monte Carlo simulation with varying mutation and crossover rates and choose the best values. Thus, the tuning of parameters in certain EAs is itself an optimization problem[60]. Then the genetic algorithm is run with more points and/or tighter tolerances on the problem. The "mysticism of numbers" artistic tuning of various methods leads to frustration when attempting to compare various methods. Reducing the need for artistic tuning was an implicit motivation for the LA approach.

Now, consideration is given to methods allowing a global search of the space, or equivalently forcing point selection near the located optimal points (i.e., a local search). This requires a mixture of global and local searching. Mathematically, this is written as

$$g(\mathbf{x}) = \alpha\, g_G(\mathbf{x}) + (1-\alpha)\, g_L(\mathbf{x}) \tag{3.16}$$

where α is the acceleration parameter and $\alpha \in [0, 1]$, g_G is the global search, and g_L is the local search. A value of α close to zero focuses on a sharp, local search for minima, while α close to one focuses on a broad, global search across all minima.

Techniques are investigated to emphasize the local search (i.e., guarantee more points in the region of the minima). The remainder of this section describes the techniques.

3.6.1 Choices of PDF

The idea in developing the LA algorithm was to have a technique that spent most of the initial effort on a global search, identifying regions of interest, and the final effort on refining the search around the found minima. The uniformity of selecting y is not mandatory and choices other than uniform random give different characteristics which will here be analyzed.

The choice of PDF from which the random values are chosen is not unique as the LA method is probabilistic (in both the N-dimensional search space, $\mathbf{x}$, and the 1-dimensional variable, y). However, the choice greatly influences the emphasis of global versus local search. Figure 3.32 shows various PDFs which emphasize local and global searches. The uniform search does not emphasize the global or the local search.

A linear function will allow the PDF to be more concentrated on the global or local search. Consider

$$PDF(y) = \frac{\gamma \left[\left(J_{\max}(\mathbf{x}) + J_{\min}(\mathbf{x}) \right) / \left(J_{\max}(\mathbf{x}) - J_{\min}(\mathbf{x}) \right) \right] + 1}{J_{\max}(\mathbf{x}) - J_{\min}(\mathbf{x})} \tag{3.17}$$

where the parameter $\gamma \in [-1, 1]$ defines what type of search is being performed. As $\gamma \to -1$, a global search is performed while as $\gamma \to 1$, a local search is performed. As $\gamma \to 0$, the search is uniform. Figure 3.33 shows some example PDF functions

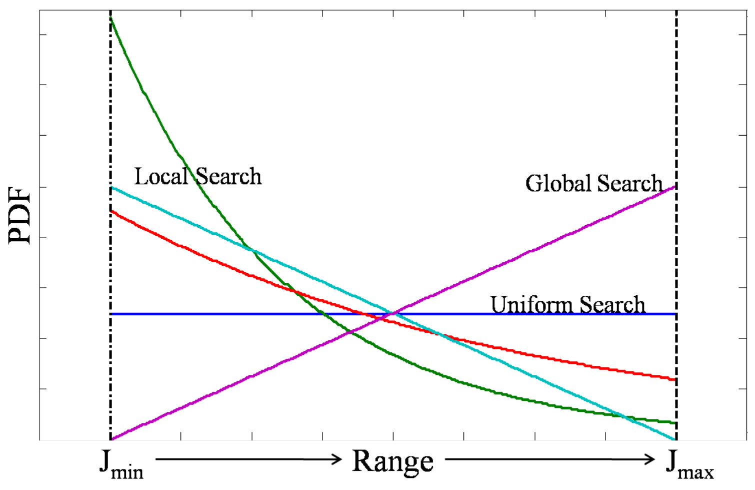

Figure 3.32: Example Probability Density Function Shapes

with arbitrarily chosen values of $J_{\min}(\mathbf{x}) = 0$ and $J_{\max}(\mathbf{x}) = 10$ for the linear PDF (on the left). Note that the integral of the PDF is equal to 1.

An exponential function will similarly allow a PDF, from which random points are chosen, to emphasize the local search or a uniform search throughout the space. Consider the function

$$PDF(y) = ke^{\frac{\gamma(1-y)}{1-\gamma}} \tag{3.18}$$

where the constant k normalizes the integral when taken from $J_{\min}(\mathbf{x})$ to $J_{\max}(\mathbf{x})$ and is written as

$$k = \frac{d}{e^{d(1-J\min(\mathbf{x}))} - e^{d(1-J\max(\mathbf{x}))}} \tag{3.19}$$

where $d = \gamma/(1-\gamma)$. The value of $\gamma \in [-1,0) \cup (0,1]$ is a tuning parameter which when $\gamma \to 0$ constitutes a uniform search and when $\gamma \to 1$ approaches a sharp, step function where only the minimum value, $J_{\min}(\mathbf{x})$, is chosen. To emphasize global searching, allow $\gamma < 0$. Example exponential functions are shown on the right in Figure 3.33.

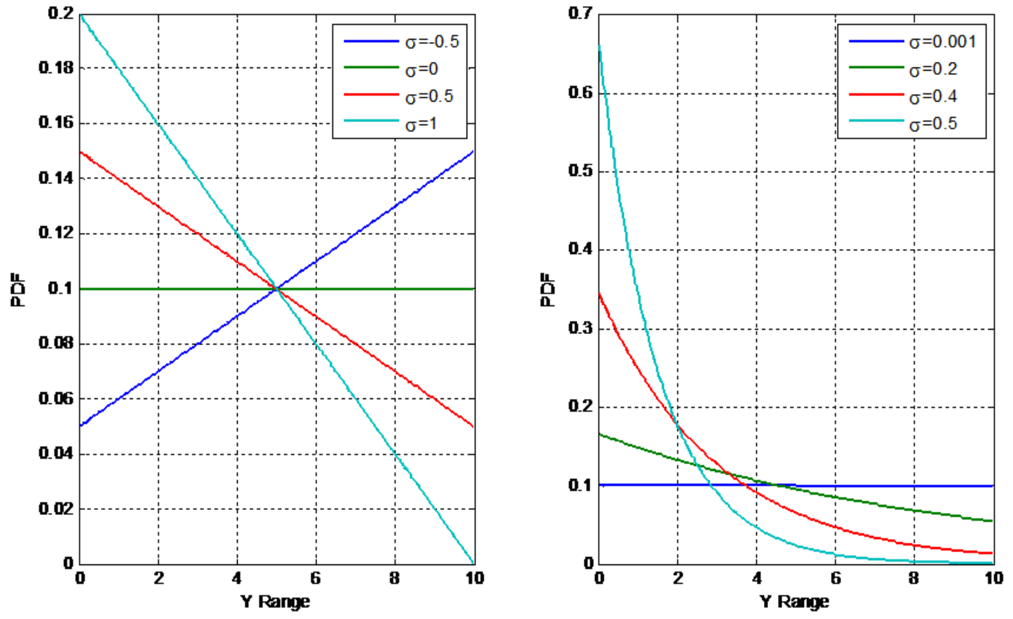

Figure 3.33: Linear and Exponential PDF Examples

As an example, the following three figures show different results using various PDFs. For this example, the sphere function, Eq. (3.5), in two dimensions is used with the bounds $-4 \leq \mathbf{x} \leq 4$. First, Fig. 3.34 shows the results using a uniform distribution. Note that the results are similar to those shown in Fig. 3.9, but the bounds are different. Next, Fig. 3.35 shows the results of a global search, meaning a linear search with $\gamma = -1$. This figure shows that there are three values near the minimum, but most of the points are spread away from the minimum. Finally, Fig. 3.36 shows the results of a steep local search, using an exponential search with $\gamma = 0.5$. This figure shows that many values are placed near the minimum, with a few points spread around the higher cost function values.

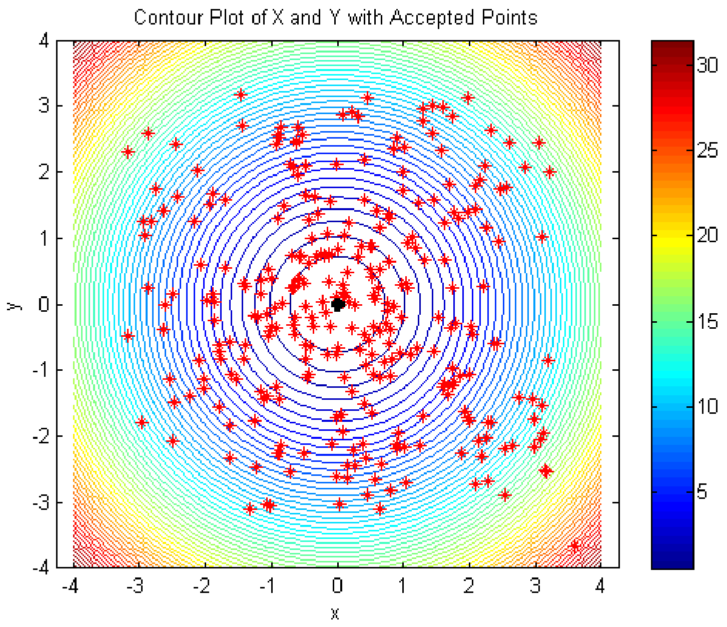

Figure 3.34: Uniform Search, $\gamma = 0$

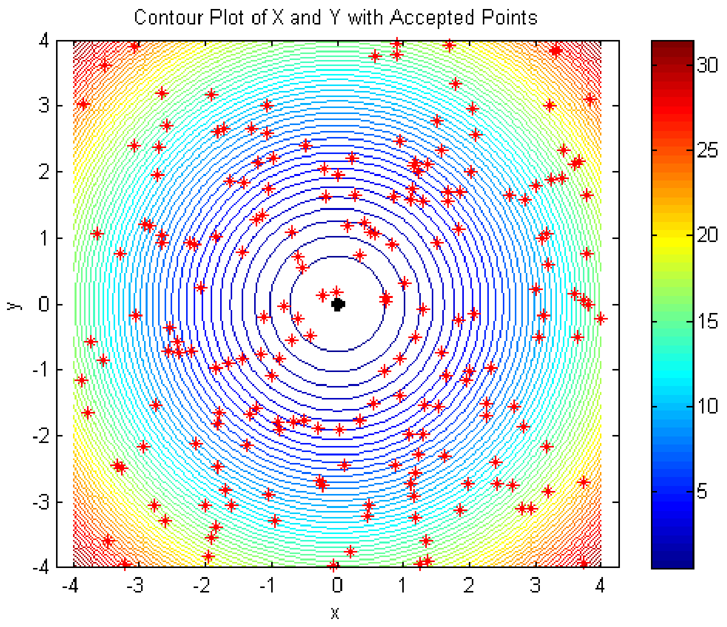

Figure 3.35: Linear Global Search, $\gamma = 1$

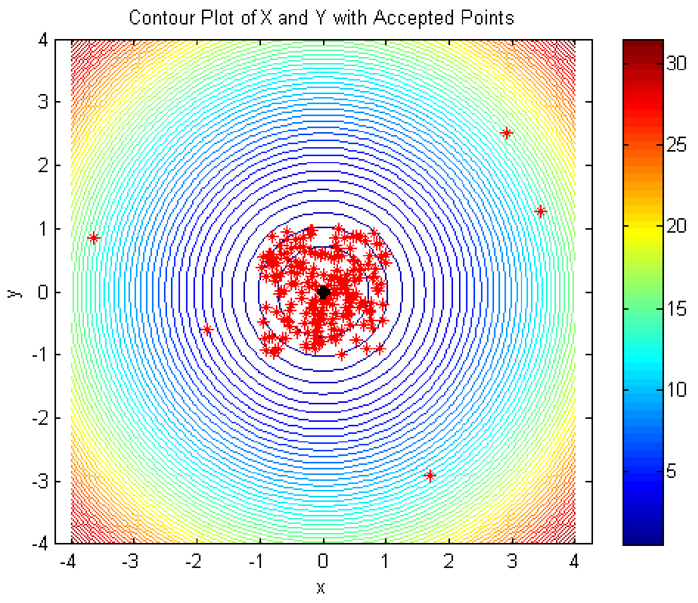

Figure 3.36: Exponential Local Search, $\gamma = 0.5$

Randomly sampling the chosen PDF function becomes quite easy as it is in one dimension. Inverse transform sampling[64] is a method that generated random numbers from a PDF based on the cumulative distribution function (CDF), subject to the constraint that the distribution is continuous and that the integral of the PDF may be inverted. This method is highly efficient when the CDF is analytically invertible (as is the case with the linear and exponential functions). The method follows: first, choose a random area (as the functions are PDFs, the area integral is bound by [0,1]). Then solve the inverse problem (of the PDF function) for the value of y that gives the selected area. Finally, use y as a random number chosen from the PDF.

3.6.2 Limiting $y_{\max}$

The simplest technique investigated is to limit the maximum value from which the random values are sampled (the limiting value could be considered a tuning parameter with a relationship to α in Eq. (3.16)). Limiting $y_{\max}$ allowed a uniform distribution to still be used but emphasized the local search.

As an example, consider the absolute value function, $J(x) = |x|$ over the range $-1 \leq x \leq 1$. Instead of allowing the value of y in Algorithm 10 to vary over the full, known range $y \in [0, 1]$, a user could limit the searched values of y to focus on the minimum, such as only allowing $y \in [0, 0.5]$. This method limits the allowable space of the point selection, forcing the minimum values to be explored with a larger number of points. However, based on probability theory, points above the line $J(x) = 0.5$ may be accepted, especially early in the optimization process when there are only a few points in the database.

On the other hand, allowing y to have a larger range such as $y \in [0, 5]$ will focus on the global search as the probability of accepting a point is much greater (as y will probabilistically be greater than or equal to $J(x) = |x|$ which will lead to more points being accepted away from the minimum value).

The downside of this method is that the probability density function (PDF) is not continuous, but instead is a step function where the value of the PDF above the chosen maximum of y is equal to zero. For this example, $J(x) = |x|$, if the user chose $y \in [0, +\infty]$ the LA effectively becomes the pure random search, assuming the x values are bound.

3.6.3 Log-Normal Distribution

Another technique is to use a log-normal distribution instead of a uniform random distribution in point selection. A log-normal distribution is a continuous distribution in which the logarithm of the random variable is normally distributed. The notation for the log-normal distribution is $\ln N(\mu, \sigma^2)$ and the probability density function is given by

$$f_X(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}, \quad x > 0 \tag{3.20}$$

where the parameters $\sigma^2 > 0$ and $\mu \in \Re$.

The values of σ and μ then become tuning parameters to force the PDF into a shape that is suitable for the user. The PDF defines the amount of global and local search in the algorithm.

In a minimization problem, this is obviously useful, and thus the log-normal distribution is implemented to force more focused searching of the minima while still exploring the entire search space with some non-zero probability. The parameters defining the log-normal distribution, σ and μ, are the tuning parameters in this case. Neither technique described in this section is implemented in the examples shown in this chapter, but they are shown to be effective in speeding up point selection near the optimal points in recent papers [54, 55].

A similar method is to take the absolute value of the normally distributed random numbers in the range of the minimum and maximum cost function values.

3.6.4 Adaptive Bounding of $J(\mathbf{x})$

The methodology of selecting *a priori* the PDF for y is a challenging subject. By tightening the bounded values of $y \in J(\mathbf{x})$ in the LA algorithm, the search emphasizes the local search. If the bounds are reduced too much, too many points are rejected, rendering the algorithm computationally inefficient. On the other hand, if the bounded values of y are too large, the global search is very broad and points are wasted. In the extreme case, the LA resembles a purely random search when the bounds on y are excessively large. Thus, an adaptive scheme is implemented as well in which the PDF changes as the number of points accepted increases. The

PDF function may even be a morphing (i.e., adaptive) function which changes from uniform, or global, search to a local search as the number of points accepted increases.

The bounds of y should be adaptively changed based on two criteria:

1. the largest and smallest cost function value of the points accepted, and

2. the number of points rejected.

If a new point has a cost function value lower than all of the points currently in the sample database, then $y_{\min}$ should be decreased to the new minimum value. Similarly, the value of $y_{\max}$ should be increased when a new point has a cost function value that exceeds the values currently in the sample database.

If the number of points rejected in the LA algorithm grows exceedingly large, this means that the algorithm is focusing on too narrow of a local search and the y range is too small. By monitoring the number of points rejected, the range of y can be adaptively changed based on the user's requirements of computational efficiency and local search capability.

3.6.5 Known and Unknown Bounds on $J(\mathbf{x})$

For the benchmark function examples given above, the bounds on y were set to exactly the minimum and maximum values of the cost function, $J(\mathbf{x})$, as the cost function is easily evaluated (especially in the 2-D cases). However, for general optimization problems, $J(\mathbf{x})$ cannot be analytically bound, especially before the optimization process. Physical bounds are implemented where and when available. For example, in engineering applications, the minimum cost function value may be zero if we know that the cost function must be positive. Thus, the minimum allowed value of y is initially set to 0 and then adaptively increased according to the number of points rejected.

When the bounds are unknown, the cost function values of the initial $2N$ points are used as the initial bounds on y. As more points are added, the range of y is adaptively modified.

For the applications in the next chapter, the bounds on y cannot be analytically determined. Therefore, an adaptive method of bounding y is implemented. The cost function values of the initial $2N$ points served as the initial bounds on y. For the first one-half of the maximum number of points (i.e., function evaluations), the upper and lower bounds are a constant multiplied by the largest and smallest cost function values, respectively, of the points selected so far in the process. In other words, if the cost function value of the $(2N+1)^{th}$ point is the largest out of the $2N+1$ points, then the bounds on y are increased to match the value of the cost function. From there, at specified intervals of the maximum allowed number of points, the maximum value is decreased in order to require a more local search as more points are accepted. This procedure made the PDF from which the values of y are chosen a series of (discontinuous) step functions.

Sanity checks are also used, limiting the number of rejected points. For example, if the number of rejected points exceeded a given value, then the value of $y_{\max}$ is increased. This made the algorithm more computationally efficient and broadened the search to a more global search while simultaneously avoiding allowing the algorithm to get stuck and not add points to the database.

3.7 Database Management and Nearest Neighbor Search

Consideration is now given to memory management and nearest neighbor search aspects of the LA algorithm. As written in Algorithm 10, the database of N-dimensional samples grows as each new sample is added. The maximum size of the database is restricted only by the user (e.g., the total number of accepted points).

Algorithm 10 does not specify the nearest neighbor search method.

3.7.1 Database Management

The first database idea implemented made the LA look more like an evolutionary algorithm and is referred to as the *cyclical database method*. The size of the database is restricted to a given number which is less than the maximum number of points. As a new point is added, an old point was removed. As an example, consider the database size to be restricted to 1,000 points, and 5000 function calls allowed to solve an optimization problem. When the 1001^{th} point is accepted, the point with index 1 is removed from the database and the rest of the points are moved up one index (i.e., index 2 becomes index 1, index 3 becomes index 2, etc.). However, if the point at index 1 has the best cost function value in the database, it is removed and put back in the database at index 1,000 and point 2 is removed. The pseudocode in Algorithm 11 shows the flow of this method.

Algorithm 11 Pseudocode of Cyclical Database Method

```
 1: Specify $K > 2N$, the number of points in the database
 2: Specify $M$, the total number of points (i.e., function calls) in the search
 3: Generate (and index) $2N$ points in the search space
 4: while Number of points $< K$ do
 5:    Generate and add point to database at next available index
 6: end while
 7: while Number of points $< M$ do
 8:    Generate a point and add to database at index=$(K + 1)$
 9:    if index=1 has best value of $J(\mathbf{x})$ then
10:       Move point from index=1 to index=$(K + 2)$
11:       Decrement all indices by 2 (i.e., index=3 becomes index=1, etc.)
12:    else
13:       Delete point at index=1
14:       Decrement all indices by 1 (i.e., index=2 becomes index=1, etc.)
15:    end if
16: end while
```

The idea behind this cyclical database method was to make memory usage more efficient, to allow the members of the population to adaptively bound y, and to gradually change from global to local search based on the dynamic population in the database. First, the memory usage became more efficient as the number of points could not grow larger than a given size (which by design is smaller than the maximum number of allowed points). This allowed for faster searches in the nearest neighbor in Step 5 of Algorithm 10. Next, the value of y was simply–yet adaptively–bounded by a constant multiplied by the best and worst cost function values in the database which broadened the range of acceptable y values. Finally, as more points are accepted, they are near the extremal points. This means that the probability of accepting a point nearer the extremal point is greater than that of accepting a point farther from the extremal point. Thus, as more points are accepted near the extremal points, the more localized the search becomes.

A different idea is to keep all of the points, but only do the nearest neighbor comparison on a subset of points taken from the database. This method is not investigated beyond recognizing it as a possibility for future research.

The most important idea behind the cyclical database method was to allow the LA to be flexible about locating the minimum. This gives the LA the possibility to *track* the minimum in time-varying systems, as discussed below.

3.7.2 Nearest Neighbor Search Techniques

The organization of the data is critical to the computational efficiency of the overall optimization process. An unorganized database will generally slow down the process significantly. The data is generally a non-uniform distribution and new samples are continuously added (and at times deleted or modified), making the database dynamic. The remainder of this section is devoted to methods of proximity searching, known as the *nearest neighbor search* (NNS).

The problem description follows: given a set S of points in a given metric space M, and a specified point, in computer science called a *query point*, $p \in M$, find the closest point in S to p. For this dissertation, S is the database of sample points where the closest point is defined by the N-dimensional Euclidean distance.

The simplest, and slowest, method of determining the nearest neighbor to a given point in a database is a brute force search where the Euclidean distance between every point is compared and the minimum is extracted. Computer scientists call this a *linear search*. The execution time is of order $O(Nd)$ where d is the cardinality of the database (i.e., number of elements in the database), S, and N is the dimensionality of S. The method is extremely robust to data clustering or other database nonlinearities. Weber, et. al., showed that the brute force method outperforms many other approaches in higher dimensional spaces[65].

The N-dimensional k-vector approach developed by Spratling and Mortari[66] is a searchless algorithm–making it extremely fast. However, it requires that the dataset be well-distributed (or uniform) and much of the efficiency is based on a pre-computed database catalog. Thus, as the k-vector approach works best on a static database, it is not pursued for this work.

Another method that stands out to handle non-uniform and dynamic datasets is Delaunay triangulation[67]. Delaunay triangulation easily handles non-uniform and dynamic datasets and is unaffected by clustering. In $\Re^2$, Delaunay triangulation takes points and generates a edges connecting the points creating triangles. In higher dimensions, Delaunay tessellation forms hyper-triangles with a similar methodology. When using Delaunay triangulations, the three vertices of a triangle are often the nearest neighbors as the method of Delaunay perfers equilateral triangles. A good estimate of the three nearest neighbors would always be the vertices of the triangle. A previous version of the LA (used by Davis[68]) required a linear interpolation forming a hyper-plane between the $N+1$ nearest neighbors in $\Re^N$. However, this is discarded for speed and it is shown in examples that the piecewise constant method–instead of the piecewise linear interpolation–presented in Algorithm 10 worked as well as the linear interpolation method and improved the computational speed of the algorithm.

A kd-tree is essentially a binary tree in which every leaf is an N-dimensional point[69, 70]. This method is extensively used in computer science as it performs well for multidimensional searches. This method partitions the space by iteratively bisecting the search into regions containing half of the points of a parent region. In general, tree methods start with a balanced tree and lose computational efficiency as points are added and/or deleted from the database. The inefficiency comes from re-balancing operations. Searches in the tree are performed by traversing the tree from the root to the proper leaf. A kd-tree can be built (with cardinality d) in order $O(d \log_2 d)$ time. A nearest neighbor search in the tree averages order $O(\log_2 d)$, or in a balanced tree is order $O(Nd^{1-1/N})$. Note that as N gets large, this approaches the brute force search. For the LA application, the kd-tree becomes difficult because it must be re-balanced often.

Grover's quantum search algorithm[71] is a probabilistic search method and gives the solution with high probability of it being the correct solution. One advantage of the quantum algorithm is that the database need not be sorted and can be non-uniform. Whereas the linear search requires order $O(Nd)$ time to search a database of cardinality d, Grover's algorithm (and all quantum algorithms) search the database in order $O(\sqrt{Nd})$ time. The quantum algorithm is asymptotically the fastest search for an unsorted database. Thus, for dimensions 4 to 16, Grover's algorithm is faster than the kd-tree (in the worst case).

There exist applications where it is acceptable to return a probabilistic good guess of the nearest neighbor. Approximate nearest neighbor algorithms do not guarantee that the actual nearest neighbor is returned in every case, but may drastically increase speed and memory efficiency. However, the actual nearest neighbor is returned in a majority of cases, depending on the dataset. The goal is to find a point (or points) whose metric distance is at most $(1 + \epsilon)$, with $\epsilon \ll 1$, times the distance to the actual nearest neighbor.

The kd-tree storage requirements approach that of a linear (i.e., brute force) search as the dimensionality of the search space gets large. In order to use a high dimensional search space (i.e. $N > 20$), a Locality Sensitive Hashing (LSH) technique can be used [72]. The LSH method performs a probabilistic reduction of dimensions by mapping data into the same bins with high probability based on some distance metric. With a properly defined family of hashing functions, LSH performs insertions, deletions, and approximate nearest neighbor searches in constant time.

For the applications presented in the next chapter, the linear search is implemented along with the cyclical database method. This ensured that a broad selection of points are evaluated, but the database would not grow to an unmanageable size.

3.8 Parallelization of the Learning Approach Algorithm

Parallelization of the LA is considered, but not implemented. A simple way to parallelize any optimization method is to split the search space, Ω into smaller sub-regions, Ω_i, and allow each processor (or core) to perform the search within a given sub-region. This would, however, require a supervisor that provided cross-talk between the sub-regions in the case of the nearest neighbor being in an adjacent sub-region. The parallelization of the LA is an area for future work.

3.9 Application to Time-Varying Systems

The Learning Approach is applied to “slowly” time-varying systems. By slowly, we assume that points are added more quickly than the system changes due to the dynamics, the cyclical database method allows for the LA to track the minimum value as it moves through time. The LA is shown to successfully track a minimum of a time-varying system[54].

4. APPLICATIONS IN ASTRODYNAMICS

This chapter will include three applications where the Learning Approach to Sampling Optimization is applied to difficult problems in astrodynamics and compared with traditional optimization techniques in terms of performance. The three problems to be investigated are:

1. N-impulse orbit transfer and rendezvous;
2. Periodic close encounters in space; and
3. A comparison of optimization methods as applied to two Cassini spacecraft trajectory models.

For each problem, a brief introduction is given, followed by the problem set up (including necessary derivations), and finally a number of examples and comparisons. Proper conclusions are given for each application.

4.1 N-Impulse Orbit Transfer and Rendezvous

4.1.1 Introduction

Consider the problem of impulsive thrust only with a constrained total time of flight. Impulsive transfers are viewed as a succession of coast arcs separated by thrust points[73]. The problem of fuel-optimal transfers has been studied by such well-known researchers as Goddard, who proposed rocket trajectories to high altitudes, and Hohmann[74], who solved the (two-impulse) minimum-fuel transfer between two circular co-planar orbits. Hohmann's transfer is generalized to elliptic orbits by Marchal[75]. The three-impulse transfer is introduced by Hoelker and Silber[76]. Other important works include those by Lawden[77], who developed primer vector theory, and Prussing[78], with a novel application to the transfer problem. Jezewski[79] developed the necessary conditions using the primer vector. These conditions were[77, 79]:

1. the primer vector and its first derivative are continuous everywhere,
2. when an impulse occurred, the primer vector has unit magnitude and is aligned with the impulse,
3. the primer vector magnitude never exceeded unity on a coast arc, and
4. the time derivative of the primer vector magnitude must be zero at all interior points separating coast arcs.

Indirect methods are proposed to solve the impulsive transfer problem[80, 81], but have the limitations of requiring the initial and final points known as well as the convergence domain generally being small. Indirect methods as those based on the calculus of variations which formulate the problem as a two-point boundary value problem (or set of problems) solved by terminal conditions. Thus, numerical optimization techniques should be employed. The GA has been thoroughly studied for optimizing orbit transfers [82, 83] and were shown to be well suited to numerically solve aspects of the orbit transfer problem, including interplanetary trajectories.

Significant work has been given to solving the minimum Δv orbit transfer and rendezvous problem for N-impulses [84, 85, 86, 87]. When $N > 2$, an analytical solution is difficult (if not impossible) to find, except in very special cases. This is due to the fact that the problems have many local minima–and generally numerous discontinuities in both the function and derivative, assuming the derivative can even be computed–which inhibit convergence to the global minimum value by traditional, calculus-based optimization methods.

A major limitation of previous work is that for $N > 3$ the orbit transfer problem becomes computationally intractable [86], resolution of the bit-encoded GA caused noticeable error, or convergence tolerances are necessarily large to make the problem solveable [85]. More importantly, only up to a three impulse transfer or two impulse rendezvous example is given in Refs. [86, 85]. The infeasible computation problem is due to the random selection of Δv values inside of the Genetic Algorithm. Another method is devised by Mortari and Henderson [88] that randomly selected radii and minimized the Δv between the radii. However, even this method becomes computationally unfeasible. Thus, this method is not further pursued. The question, then, became how to analytically bound the Δv components such that the N-impulse problem is solved.

This section introduces new, novel constraints on the N-impulse selection which strongly bound the size of the search space, Ω. This reduction makes the N-impulses problem affordable for most of the algorithms reviewed in Chapter II as well as the Learning Approach presented in Chapter III.

The outline of this section follows: first a presentation of limitations on the velocity that guarantee that after each impulse, the new orbit does not intersect the Earth (plus altitude constraint) is given. This allows the choice of velocity components to be bounded. The algorithm for orbit transfer and rendezvous is then developed. Finally, this section compares the performance of the GA and LA optimization techniques in terms of performance for a set of examples using real spacecraft. Orbit transfers using $N > 5$ impulses is shown to be easily computed by a the LA and GA when the velocity constraints are used.

4.1.2 Δv Limitations

In order to find the minimum Δv solution in a more efficient manner, either the dimension is decreased, or the search space is more tightly constrained. As the problem to be solved involves constructing the method to solve the N-impulse problem, the dimensionality of the problem cannot be decreased and thus the search space must be constrained. Here, the limitations on the impulse, Δv, in order to obtain an orbit whose perigee radius is greater than the minimum given value, r_p, are analyzed. The value r_p represents a minimum radius from the Earth that the spacecraft is allowed to obtain. In order to derive the analytical bounds for this problem, the impulsive Δv is split into the following, generally non-orthogonal, three components:

$$\Delta \mathbf{v} = \Delta v_r \, \hat{\mathbf{r}} + \Delta v_v \, \hat{\mathbf{v}} + \Delta v_h \, \hat{\mathbf{h}} \tag{4.1}$$

where $\hat{\mathbf{r}}$, $\hat{\mathbf{v}}$, and $\hat{\mathbf{h}}$, are the unit-vector directions of radial, velocity, and angular momentum components, respectively. The Δv components appearing in Eq. (4.1) are bounded as follows and the bounds are derived.

$$\Delta v_{r\,\min} \le \Delta v_r \le \Delta v_{r\,\max} \tag{4.2}$$

$$\Delta v_{v\,\min} \le \Delta v_v \le \Delta v_{v\,\max} \tag{4.3}$$

$$\Delta v_{h\,\min} \le \Delta v_h \le \Delta v_{h\,\max} \tag{4.4}$$

Let $\mathbf{r}$ and $\mathbf{v}$ the position and velocity of an Earth-orbiting satellite at a given time, t. Classical Keplerian motion gives the following expression for angular momentum

$$\mathbf{h} = \mathbf{r} \times \mathbf{v} = h \, \hat{\mathbf{h}} = \sqrt{\mu \, p} \, \hat{\mathbf{h}} \tag{4.5}$$

where h is the angular momentum modulus, μ the gravitational constant of Earth, and p the semi-latus rectum. Equation (4.5) leads to $p = h^2/\mu$, while the energy equation

$$\frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2\,a} \tag{4.6}$$

gives an expression for the semi-major axis, $a = \mu/(2\mu/r - v^2)$. Additional relationships that are used in the development are

$$p = a(1 - e^2), \qquad e = \sqrt{1 - \frac{p}{a}}, \qquad \text{and} \qquad r_p = a(1 - e) \tag{4.7}$$

relating the eccentricity, e, and the perigee radius, r_p. For further definition of the orbital parameters, see the Section entitled *Brief Review of Astrodynamics* in Chapter II or one of the many excellent texts on orbital mechanics such as Refs. [50, 48, 49]. In the following derivations, it is assumed that the orbit of the satellite is elliptical (i.e. $a > 0$ and $0 \leq e < 1$). This is equivalent to a statement about the orbit energy, $r\,v^2 < 2\mu$.

4.1.2.1 Limitation on $\Delta v_{v\,\max}$

It is intuitive that any increase along the velocity vector direction does not decrease the radius of perigee, but will, in general, increase the perigee altitude. The worst case occurs when the impulse is applied exactly at the perigee, in which case the radius of perigee will remain the same after a positive Δv_v but the semi-major axis and eccentricity will increase.

A short mathematical proof is here provided that shows $\Delta v_{v\,\max}$ is unbounded (in practice, the value is bounded by the user as shown later). The proof investigates what happens to r_p, the radius of perigee, when the velocity vector is increased by an infinitesimal amount. The new velocity can be written as

$$\mathbf{v}' = \lambda\,\mathbf{v} = (1+\varepsilon)\,\mathbf{v} \tag{4.8}$$

where $0 < \varepsilon \ll 1$ and the $(\,)'$ notation identifies the value after the infinitesimal impulse. Only the first order approximation of the perturbed orbital parameters as a function of the original parameters and the small variation (i.e., differential) ε are considered.

It is clear from Eq. (4.5) that $p' = \lambda^2\,p$ and then

$$p' \approx (1+2\varepsilon)\,p \tag{4.9}$$

Equation (4.6) is used to obtain an expression for the ratio a'/a

$$\frac{a'}{a} = \frac{2\mu/r - v^2}{2\mu/r - \lambda^2\,v^2} \approx \frac{1}{1 - 2\varepsilon\,v^2\,a/\mu} \approx 1 + \frac{2v^2\,a}{\mu}\,\varepsilon$$

from which is obtained

$$a' \approx a\,\left(1 + \frac{2\,v^2\,a}{\mu}\,\varepsilon\right) \tag{4.10}$$

Using Eqs. (4.9), (4.10), and (4.7), the following expression is derived

$$\frac{p'}{a'} \approx \frac{p}{a}\left(\frac{1+2\varepsilon}{1+2v^2a\varepsilon/\mu}\right) \approx \frac{p}{a}\left[1 + \left(2 - \frac{2v^2a}{\mu}\right)\varepsilon\right]$$

which gives

$$(e')^2 = 1 - \frac{p'}{a'} \approx \left[1 - \frac{p/a}{1-p/a}\left(1 - \frac{v^2a}{\mu}\right)\varepsilon\right] \tag{4.11}$$

which implies

$$1 - e' \approx (1-e)\left[1 + \frac{e}{1-e}\frac{p/a}{1-p/a}\left(1 - \frac{v^2 a}{\mu}\right)\varepsilon\right]$$

Finally [after simplification using Eqs. (4.6) and (4.7)], the expression for r_p' is obtained

$$r_p' \approx r_p\left[1 + \frac{2}{e}\left(1 - \frac{r_p}{r}\right)\varepsilon\right] \tag{4.12}$$

It is clear from Eq. (4.12) that as long as $r > r_p$ (i.e., not at the perigee), then $r_p' > r_p$. Obviously, when $r = r_p$ the velocity vector, $\mathbf{v}$, is orthogonal to the position vector, $\mathbf{r}$, and the perturbed velocity, $\mathbf{v}'$, does not modify the perigee. Thus, $\Delta v_{v\,\mathrm{max}}$ cannot be analytically bounded as expected, but will however, be practically bounded based on operation constraints.

4.1.2.2 Limitation on $\Delta v_{v\,\mathrm{min}}$

Now, the limitation on $\Delta v_{v\,\mathrm{min}}$ is derived. Setting $v' = v - \Delta v$ leads to an expression for the semi-major axis, a, as a function of Δv from the energy equation

$$\frac{(v')^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2\,a'} \qquad \to \qquad a' = \frac{\mu\, r}{2\mu - r\,(v')^2} \tag{4.13}$$

while the angular momentum allows ($\mathbf{v}'$ and $\mathbf{v}$ are parallel at the instant of the impulse)

$$h' = h\,\frac{v'}{v} = \sqrt{\mu\, a(1-(e')^2)} \tag{4.14}$$

Note that $r = r'$ as the maneuver is impulsive. An expression for e' as a function of Δv is then obtained

$$\frac{h^2\,(v')^2}{v^2} = \mu\, a'(1-(e')^2) = \frac{\mu^2\, r\,(1-(e')^2)}{2\mu - r\,(v')^2} \tag{4.15}$$

and

$$e' = \frac{1}{\mu\, v}\sqrt{\frac{\mu^2\, r\, v^2 - h^2 (v')^2\,(2\mu - r\,(v')^2)}{r}} \tag{4.16}$$

Hence, the Δv associated with the new minimum perigee, $r_p = a'(1-e')$, is

$$r_p = \frac{\mu\, r}{2\mu - r\,(v')^2}\left(1 - \frac{1}{\mu\, v}\sqrt{\frac{\mu^2\, r\, v^2 - h^2 (v')^2\,(2\mu - r\,(v')^2)}{r}}\right) \tag{4.17}$$

Solving Eq. (4.17) for v', we obtain

$$\left[r_p^2\, v^2\,(2\mu - r\,(v')^2) + r\, h^2\,(v')^2 - 2r\, v^2\, r_p\, \mu\right](2\mu - r\,(v')^2) = 0 \tag{4.18}$$

from which the trivial solutions (associated with hyperbolic solutions) are obtained

$$\Delta v_v = v \pm \sqrt{\frac{2\mu}{r}} \tag{4.19}$$

and the two searched equations

$$\Delta v_{v\,\min} = v \pm v\sqrt{\frac{2\mu r_p (r - r_p)}{r(h^2 - r_p^2 v^2)}} \tag{4.20}$$

Using Eq. (4.20), a numerical test was performed to prove the correctness of the equation. The orbit was selected has classical Keplerian orbit elements shown in Table 4.1.

Table 4.1: Orbital Elements Used for $\Delta v_{v\,\min}$ Test

a	e	i	ω	Ω	φ
8000 km	0.05	0 deg	0 deg	0 deg	0 deg

The constraint of minimum allowed altitude was 350 km. For the test, 1,000 impulsive velocity changes using Eq. (4.20) are applied equally spaced around the orbit (in terms of true anomaly). The resulting difference in computed and constrained radius of perigee are plotted in Figs. 4.1 and 4.2 for the positive and negative values of Δv_v, respectively.

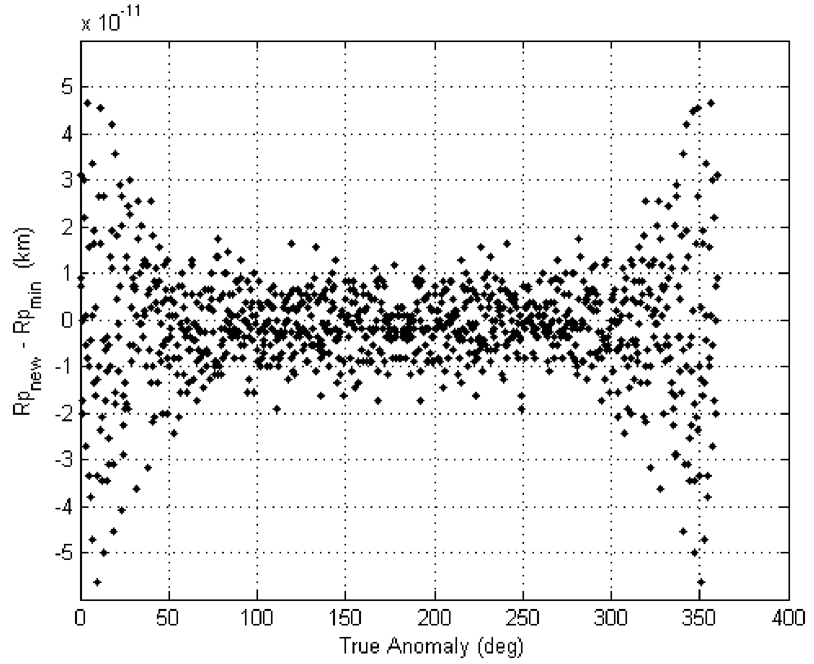

Figure 4.1: Results of Validation Tests for $\Delta v_v > 0$

4.1.2.3 Limitation on Δv_r

Now, the effects of impulses along the radial direction are derived. Radial impulses do not change the angular momentum vector as

$$\mathbf{h}' = \mathbf{r} \times (\mathbf{v} + \Delta v_r \, \hat{\mathbf{r}}) = \mathbf{r} \times \mathbf{v} = \mathbf{h} \tag{4.21}$$

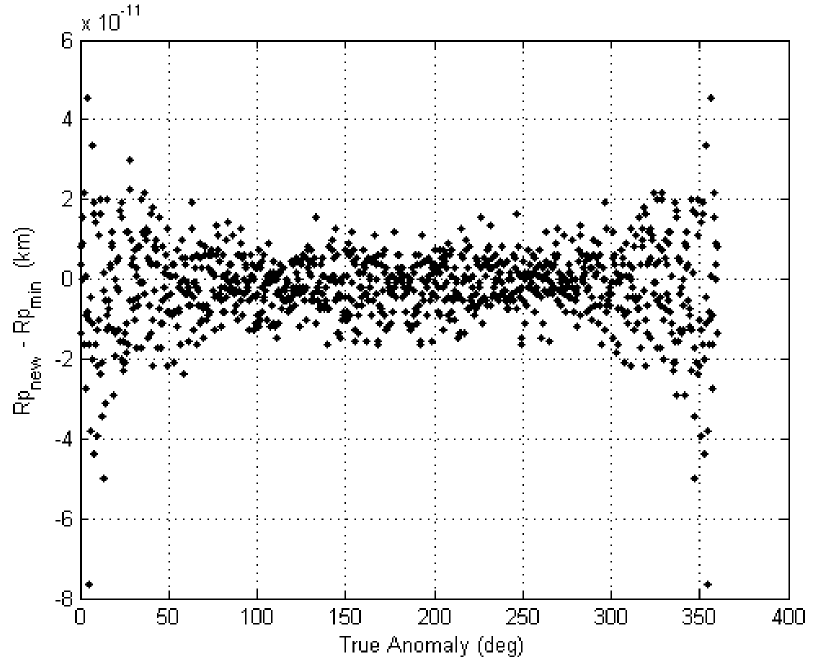

Figure 4.2: Results of Validation Tests for $-\Delta v_v < 0$

The energy equation, Eq. (4.6), becomes

$$\frac{v^2 + \Delta v^2 + 2\,\Delta v\,\mathbf{v}\cdot\hat{\mathbf{r}}}{2} - \frac{\mu}{r} = \frac{v^2 + \Delta v^2 + 2\,v\,\Delta v \cos\alpha}{2} - \frac{\mu}{r} = -\frac{\mu}{2\,a'} \qquad (4.22)$$

where α is the angle between the radius and velocity vectors. From the above equation, an expression for a' is computed as

$$a' = \frac{\mu\, r}{2\mu - r\,(v^2 + \Delta v^2 + 2v\Delta v \cos\alpha)} \qquad (4.23)$$

The angular momentum equation, as written in Eqs. (4.5) and (4.7), allows an expression for e as a function of Δv to be obtained

$$h^2 = \mu\, a'(1 - (e')^2) = \frac{\mu^2\, r\,(1 - (e')^2)}{2\mu - r\,(v^2 + \Delta v^2 + 2v\Delta v \cos\alpha)} \qquad (4.24)$$

The value of Δv associated with minimum perigee is then

$$r_p = \frac{\mu\, r}{2\mu - r\,(v^2 + \Delta v^2 + 2v\Delta v\cos\alpha)} \left(1 - \sqrt{1 - \frac{h^2[2\mu - r\,(v^2 + \Delta v^2 + 2v\Delta v\cos\alpha)]}{\mu^2\, r}}\right) \tag{4.25}$$

Setting $x = 2\mu - r\,(v^2 + \Delta v^2 + 2v\Delta v\cos\alpha)$, Eq. (4.25) is written as

$$x\,(r_p^2\, x - 2\, r_p\, \mu\, r + h^2\, r) = 0 \tag{4.26}$$

Solving the equation for the non-trivial solution gives

$$\Delta v_r^2 + 2v\,\Delta v_r\, \cos\alpha + v^2 - \frac{2\mu}{r} + \frac{2\,\mu}{r_p} - \frac{h^2}{r_p^2} = 0 \tag{4.27}$$

which has solutions

$$\Delta v_r = -\sqrt{\frac{\mu}{p}}\, e\, \sin\varphi \pm \sqrt{\left(\frac{1}{r} - \frac{1}{r_p}\right)\left[2\mu - h^2\left(\frac{1}{r} + \frac{1}{r_p}\right)\right]} \tag{4.28}$$

where φ is the true anomaly.

Again, an orbit, whose elements are shown in Table 4.1, is generated with the minimum altitude constraint of 350 km. For the test, 1,000 impulsive velocity changes along the radial direction (both positive and negative) are applied equally spaced around one orbit (in terms of true anomaly) according to Eq. (4.28). The resulting difference in computed and constrained radius of perigee is plotted are given in Figs. 4.3 and 4.4 for the case of positive and negative Δv_r, respectively.

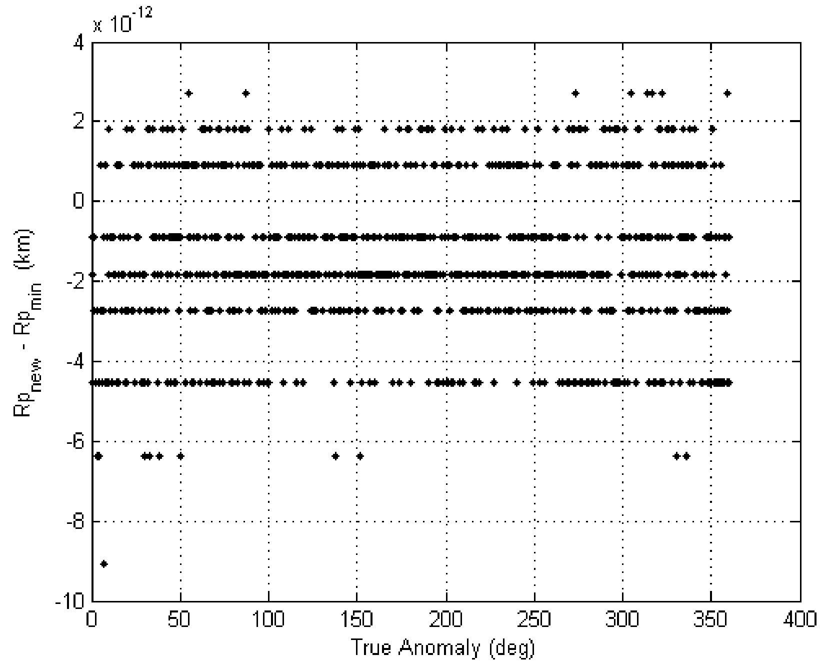

Figure 4.3: Results of Validation Tests for $\Delta v_r > 0$

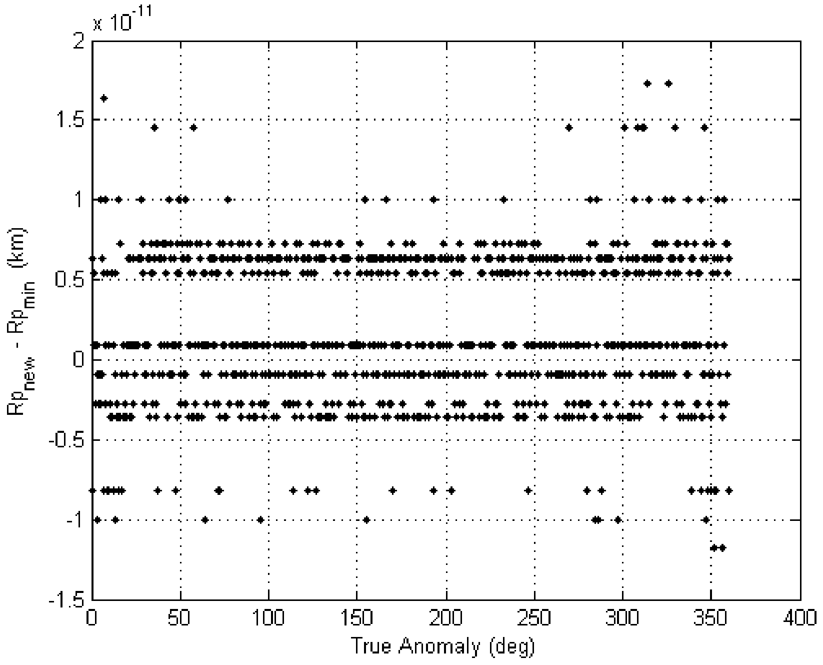

Figure 4.4: Results of Validation Tests for $\Delta v_r < 0$

4.1.2.4 Limitation on Δv_h

It is shown, via analytical proof, that neither positive nor negative Δv_h decrease the perigee altitude. In particular, the effects of Δv_h on r_p are investigated when the velocity, $\mathbf{v}$, is changed by $\mathbf{v}' = \mathbf{v} + \lambda\,\mathbf{h}/r$ with $0 < \|\lambda\| \ll 1$. Note that λ can be a positive or negative value. Let α denote the angle between $\mathbf{r}$ and $\mathbf{v}$. Since $\mathbf{v}$ and $\mathbf{h}$ are orthogonal,

$$v' = v\,\sqrt{1+\lambda^2\sin^2\alpha} \approx v\,\left(1+\frac{\lambda^2\sin^2\alpha}{2}\right). \tag{4.29}$$

Similarly,

$$h' = h\,\sqrt{1+\lambda^2} \approx h\,\left(1+\frac{\lambda^2}{2}\right). \tag{4.30}$$

Now the ratio r_p'/r_p is computed following the same procedure as previously used

$$p' = \frac{h'}{\mu} \approx \frac{h}{\mu}\left(1+\frac{\lambda^2}{2}\right)^2 \approx p\,\left(1+\lambda^2\right) \tag{4.31}$$

then

$$a' \approx a\,\left(1+\lambda^2\,\sin^2\alpha\,\frac{v^2 a}{\mu}\right) \tag{4.32}$$

and

$$1-e' \approx (1-e)\,\left[1+\frac{e}{1-e}\frac{p/a}{1-p/a}\left(1-\frac{v^2 a\sin^2\theta}{\mu}\right)\frac{\lambda^2}{2}\right] \tag{4.33}$$

yielding to

$$r_p' \approx r_p\left\{1+\frac{\lambda^2}{2e}\left[(1-e)\,\sin^2\alpha\left(\frac{2a}{r}-1\right)+(1+e)\right]\right\} \tag{4.34}$$

It is clear from this equation that for any infinitesimal perturbation (positive or negative) in the direction of the angular momentum, $\mathbf{h}$, the radius of perigee can not decrease.

Note that the bounds computed are not independent. It is clear that if we select a value for Δv_v (e.g., defined by one gene of the chromosome), then the bounds for $\Delta v_{r\,\min}$ and $\Delta v_{r\,\max}$ have to be computed using the new velocity vector $\mathbf{v}_i' = \mathbf{v}_i + \Delta v_v\,\hat{\mathbf{v}}_i$, rather than $\mathbf{v}_i$.

4.1.3 Orbit Transfer Problem

The orbit transfer and orbit rendezvous problems are set up in much the same way. In previous work, Abdelkhalik [86] used the true anomaly of the initial and final orbits and the flight time as variables that are input into a Lambert solver [48] for each transfer leg. The cost function minimized was simply the sum of the required impulsive Δv values for the entire maneuver.

For the current work, the Lambert solver is used only to compute the velocities for the final transfer leg (i.e., the final two impulses). The previous $N-2$ impulses are generated using the derived constraints presented. The variables required are the times of each impulse (N variables), the Δv magnitude of each component for the first $N-2$ impulses in the sequence ($3(N-2)$ variables), and the final orbit true anomaly (1 variable). This means for N impulses, there are $4N-5$ total variables.

For the special case of $N = 2$, the three variables are the (two) times of the impulses and the final orbit true anomaly, which means the derived constraints are not used, but the Lambert solver provides the solution based on the chosen times of impulses and anomaly. Table 4.2 lists the variables for a 3-impulse maneuver, where the Δv components are for the first impulse and represent the fraction of magnitude between the derived upper and lower bounds allowed. The extension to determine the variables with a larger number of impulses is then easily made.

All of the variables are normalized such that the values are in $[0,1]$. The time is normalized by a maximum time of maneuver supplied by the user. This allows

Table 4.2: Variables for 3-Impulses

x_1	x_2	x_3	x_4	x_5	x_6	x_7
t_1	t_2	t_3	Δv_v	Δv_r	Δv_h	φ_f

a guarantee that $t_{i+1} > t_i$. The variables associated with velocities are the fraction between the maximum and minimum value bound by the constraints derived in Eqs. (4.20) and (4.28), and a maximum allowed Δv value per impulse. The final variable is a fraction of 2π which represents the final orbit true anomaly.

The basic flow of the solution method for the orbit transfer is shown in Algorithm 12.

Algorithm 12 Orbit Transfer/Rendezvous Pseudo-code

```
1: Given initial and final orbits, $N$, $r_p$, $t_{\max}$, and $\Delta v_{\max}$
2: Generate $t_1$ and $t_i = t_{i-1} + x$, $i = 2, ..., N$
3: Generate components of $\Delta \mathbf{v}_1$
4: Propagate orbit
5: for $i = 2$ to $N - 2$ do
6:     Generate components of $\Delta \mathbf{v}_i$
7:     Propagate orbit
8: end for
9: Generate $\vartheta_f$
10: Solve Lambert problem
11: Compute $\Delta v_{total}$
```

For the examples shown in the remainder of this section, the parameters set in the GA are shown in Table 4.3. While these values may be sub-optimal for the examples presented, they are based on practices recommended by De Jong[89] and used by Kim[85], thus allowing a better comparison with previous work.

Table 4.3: Genetic Algorithm Parameters

Parameter	Selected Value
Population Size	150
Generations	100
Crossover Probability	0.8
Elite Count	2

The LA is run keeping all of the points in the database and adaptively changing the bounds of $J(\mathbf{x})$. For the first 50% of the points added to the database, the minimum and maximum cost function values are used as the bound. From there, the maximum bound of $J(\mathbf{x})$ is decreased incrementally to one-half of the maximum value in the database. This allowed the LA to have a broad, global search for the first half of the test, while slowly focusing on a local search during the second-half of the test.

4.1.3.1 Problem 1: Hohmann Transfer

The new, constrained algorithm is first verified by solving two problems with known solutions. The first problem investigated was a Hohmann transfer[74] in Earth orbit from low Earth orbit (LEO) to a geosynchronous orbit (GEO). A satellite in an equatorial, circular orbit with radius $r_1 = 7,000$ km was transferred to an equatorial, circular orbit with radius $r_2 = 42,164$ km (GEO). The Hohmann transfer requires two impulses

$$\Delta v_1 = \sqrt{\frac{\mu}{r_1}}\left(\sqrt{\frac{2r_2}{r_1+r_2}} - 1\right) \qquad \text{and} \qquad \Delta v_2 = \sqrt{\frac{\mu}{r_2}}\left(1 - \sqrt{\frac{2r_1}{r_1+r_2}}\right) \tag{4.35}$$

and the total required impulsive Δv for the Hohmann transfer is then simply $\Delta v = \Delta v_1 + \Delta v_2$.

For this example, the analytical solution is $\Delta v_1 = 2.3368$ km/s and $\Delta v_2 = 1.4339$ km/s, giving a total $\Delta v = 3.7707$ km/s. The initial optimization result by the LA provided a solution of $\Delta v = 3.7719$ km/s while the GA provided a solution of $\Delta v = 3.7783$ km/s. In this case, the GA exited well before the total number of generations is reached due to the cost function tolerance between generations being met. In total, only 12 generations were used. This shows that the GA became attracted to a local minima that lies near the correct solution.

The LA then used the same number of function evaluations to make a fair comparison. The LA outperformed the GA for the Hohmann transfer and a GA reported in previous work by Kim [85] ($\Delta v = 3.774$ km/s). As another verification, the angle between the position vectors where the impulses took place were computed and found to be 179.98° for the LA and 178.75° for the GA. Figure 4.5 shows the orbit transfer.

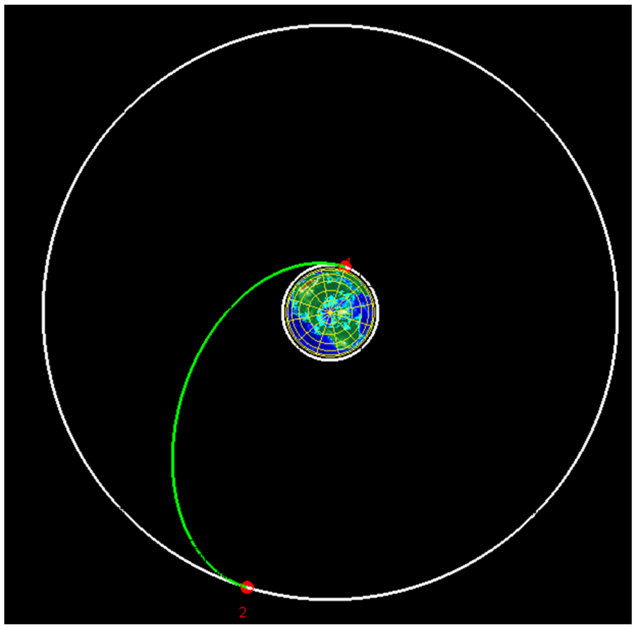

Figure 4.5: Representation of Hohmann Transfer

A Monte Carlo test is performed where the GA and LA are each run 100 times (independent of the previous results and with the same number of points, with the GA determining the number of points). Figures 4.6 and 4.7 show the results. Clearly, the LA outperformed the GA on the whole.

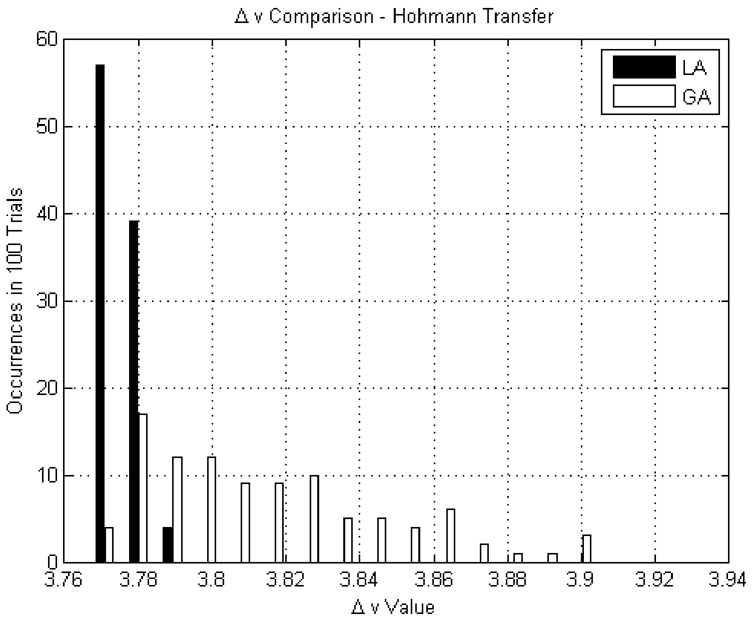

Figure 4.6: Monte Carlo Hohmann Transfer Δv Results

In addition, to prove the validity of the transfer algorithm, a test (using the LA only with 2,000 function calls) is performed where 3 and 4-impulses are given and the time is constrained to 0.75 days–roughly twice that of the actual Hohmann transfer. Given that the Hohmann transfer is optimal for the 2-impulse case, the 3 and 4-impulse cases should show one or two impulses with near zero Δv (i.e., still use the 2-impulse transfer). As can be seen in Figs. 4.8 and 4.9, and in Tables 4.4 and 4.5, this is indeed the case as some of the Δv values are indeed very small exhibiting

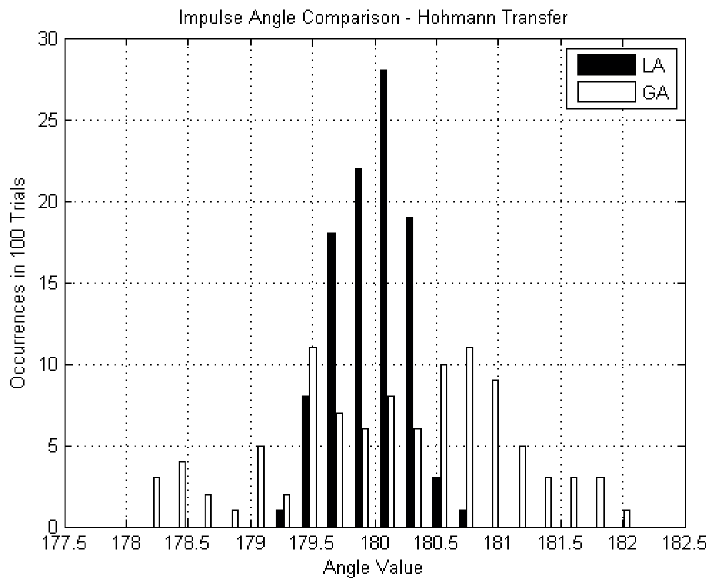

Figure 4.7: Monte Carlo Hohmann Transfer Angular Results

a Hohmann-like transfer. Note that Table 4.4 has a total velocity of $\Delta v = 3.7708$ km/s. This verifies that the orbit transfer algorithm will not necessarily use all of the allowed impulses and yields the optimal transfer.

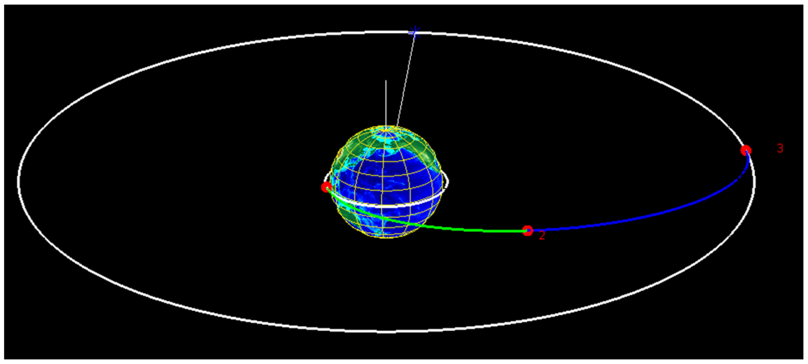

Figure 4.8: 3-Impulse Representation of Hohmann Transfer

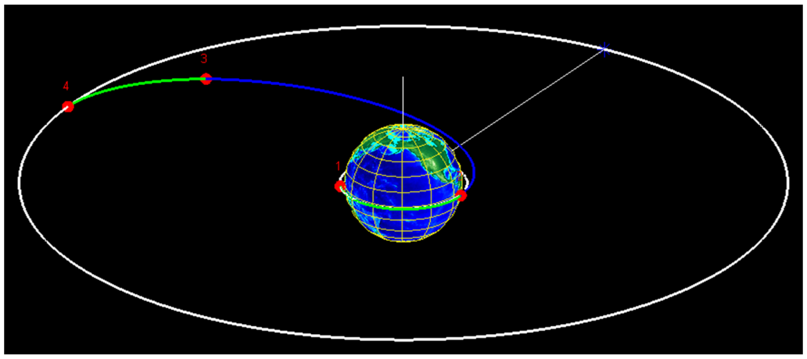

Figure 4.9: 4-Impulse Representation of Hohmann Transfer

Table 4.4: 3-Impulse Hohmann-like Transfer Parameters

$\|\Delta v_1\| = 2.3368$ km/s	Time (UTC): 19-Dec-2008, 08:56:09		
Position (km)	$R_x = -2,007.4$	$R_y = 6,706.0$	$R_z = 0$
Velocity (km/s)	$V_x = -7.2291$	$V_y = -2.1640$	$V_z = 0$
Δv (km/s)	$\Delta v_r = -0.0093$	$\Delta v_v = 2.3368$	$\Delta v_h = -0.0013$

$\|\Delta v_2\| = 1.7 \cdot 10^{-6}$ km/s	Time (UTC): 19-Dec-2008, 10:04:59		
Position (km)	$R_x = -12,422.0$	$R_y = -17,203.1$	$R_z = -2.2$
Velocity (km/s)	$V_x = 0.72597$	$V_y = -4.5638$	$V_z = -7.9 \cdot 10^{-5}$
Δv (km/s)	$\Delta v_r = 7.9 \cdot 10^{-7}$	$\Delta v_v = -1.4 \cdot 10^{-6}$	$\Delta v_h = 4.9 \cdot 10^{-7}$

$\|\Delta v_3\| = 1.434$ km/s	Time (UTC): 19-Dec-2008, 14:14:27		
Position (km)	$R_x = 12,054.1$	$R_y = -40,404.2$	$R_z = 2.1 \cdot 10^{-10}$
Velocity (km/s)	$V_x = 1.576$	$V_y = 0.4565$	$V_z = 0.0002$
Δv (km/s)	$\Delta v_r = -0.0131$	$\Delta v_v = 1.4339$	$\Delta v_h = -0.0004$

Table 4.5: 4-Impulse Hohmann-like Transfer Parameters

Impulse	$\|\Delta v\|$ (km/s)
1	0.0137
2	2.3425
3	0.0216
4	1.3946
Total	3.7724

4.1.3.2 Problem 2: Inclined 2-Impulse Transfer

In his text, Vallado[49], presented a more challenging 2-impulse problem involving orbit transfer combined with a plane (or inclination) change. Consider a satellite in a circular orbit with initial radius $r_1 = 6,671.53$ km and inclination $i = 28.5°$ (equivalent to launching from Cape Canaveral). The final orbit is circular with radius $r_2 = 26,558.56$ km and $i = 0°$. If the entire plane change is performed at r_2, then the total Δv for the maneuver required is 4.2120 km/s. However, if the first impulse includes an inclination change of 3.305° and the second impulse takes the remainder of the plane change, the total required Δv for the maneuver becomes 4.05897 km/s. The transfer is represented in Fig. 4.10.

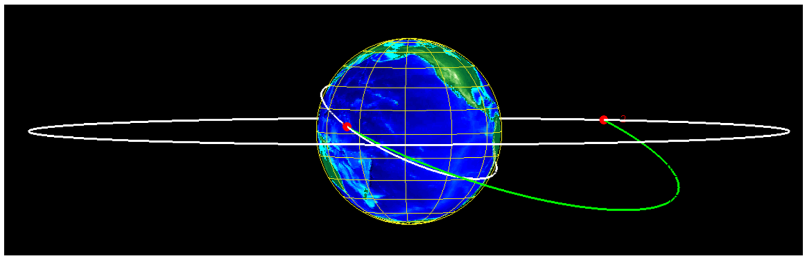
Figure 4.10: Vallado's 2-Impulse, Inclined Transfer Problem

For this example, the initial run of the LA computed a value of 4.0591 km/s and the GA computed a value of 4.0624 km/s. A Monte Carlo test is performed where 100 tests are run. Table 4.6 shows that the LA outperformed the GA in terms of accuracy for the same number of function calls. Again, the GA only used on average 14 generations (2,100 points) and therefore the LA used the same number of points. Note the standard deviation of the LA is more than three times less than the GA, showing that the LA gives a more accurate and repeatable solution.

Table 4.6: Results of Monte Carlo Test on Vallado's 2-Impulse Problem

Method	Mean Δv (km/s)	1σ of Δv (km/s)
GA	4.0627	0.0039
LA	4.0593	0.0012

4.1.3.3 Problem 3: Inclined 6-Impulse Transfer

This problem doubled the number of allowed impulses compared to previous publications [85, 86], and shows the power of the N-impulse admissible constraints in allowing high values of N. The example shown is a transfer from an orbit with $r_1 = 8,000$ km and $i = 45°$ to a GEO orbit and is represented in Fig. 4.11.

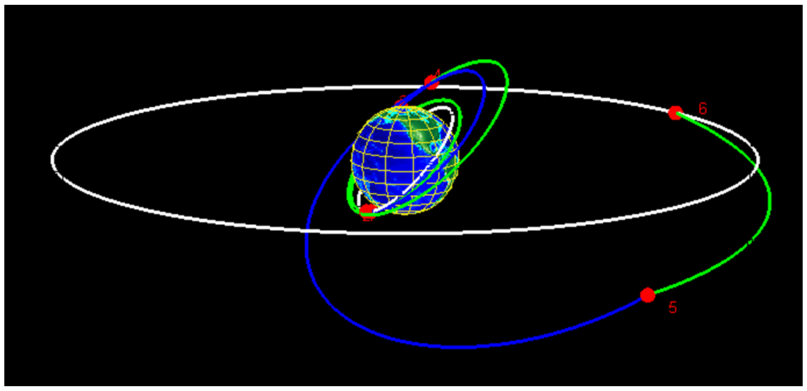

Figure 4.11: Representation of 6-Impulse Transfer

In order to obtain a feasible solution in the larger dimensional search space, more points are used. The GA is modified to use 200 generations with a population of 400. Not all of the generations are used, however, as the GA exited because the minimum function value changed less than the tolerance generally before 200 generations are used. The LA then used the same number of points as the GA in order to make a fair comparison. In a Monte Carlo test of 100 trials, the LA found an average minimum Δv value of 4.3237 km/s while the GA found an average minimum Δv value of 4.3482 km/s.

4.1.4 Orbit Rendezvous Problem

The orbit transfer and orbit rendezvous problems are purposefully set up to be very similar, where the only difference is that in the rendezvous problem, the phasing of the orbits matter in order that the chase spacecraft meet with the target spacecraft (position and velocity at a given time are the same). Previous work by Kim [85] used the magnitude of the Δv, true anomaly at the time of the maneuver, and the direction of the vector impulse as variables. The cost function used by Kim is a weighted sum of matching position and velocity components of the two spacecraft in the final orbit where the weight values are set by trial and error experimentation (Kim allowed a 1% error in position and velocity as stopping criteria). Kim's examples included only co-planar orbits with a 2-impulse orbit rendezvous (with an augmented cost function to include minimizing the Δv along with the position and velocity component errors).

For this application, the orbit rendezvous problem has the same variables as the transfer problem (time of each impulse, Δv fraction based on derived constraints, and final orbit true anomaly). The requirement in the rendezvous problem is that the chase spacecraft arrives in the final orbit at a given time with the same position and velocity vectors as the target spacecraft. This is derived and used in the Lambert solver, and therefore neither weighted nor included in the cost function. The cost function for the rendezvous problem is simply the total Δv, which we wish to minimize. The basic flow of the solution method for the orbit rendezvous is the same as the orbit transfer problem (Algorithm 12) with the only difference of propagating the target spacecraft to find the final true anomaly value. This makes the software package simple, as only a switch is needed to perform either transfer or rendezvous.

4.1.4.1 Hohmann Rendezvous

The orbit rendezvous problem is verified using a Hohmann transfer. The elapsed time before the first impulse, Δv value, and angular separation between the impulses matched the analytical values very well (the Δv is accurate below the m/s level) when using the LA with 1,500 points. Table 4.7 shows the results of the rendezvous.

Table 4.7: Hohmann Rendezvous Parameters

$\|\Delta v_1\| = 2.3368$ km/s	Time (UTC): 19-Dec-2008, 06:07:58		
Position (km)	$R_x = -6,424.6$	$R_y = -2,779.4$	$R_z = 0$
Velocity (km/s)	$V_x = 2.9962$	$V_y = -6.9257$	$V_z = 0$
Δv (km/s)	$\Delta v_r = 6.8 \cdot 10^{-6}$	$\Delta v_t = 2.3368$	$\Delta v_n = 0$

$\|\Delta v_2\| = 1.4339$ km/s	Time (UTC): 19-Dec-2008, 11:27:36		
Position (km)	$R_x = 38,697.9$	$R_y = 16,741.5$	$R_z = 0$
Velocity (km/s)	$V_x = -0.65147$	$V_y = 1.5059$	$V_z = 0$
Δv (km/s)	$\Delta v_r = 5.9 \cdot 10^{-6}$	$\Delta v_t = 1.4339$	$\Delta v_n = 0$

4.1.4.2 N-Impulse Rendezvous Examples

The GA is only compared for one example: the 4-impulse rendezvous. A true value is not known for the general problem. Figures 4.12 to 4.14 show the results of testing the LA on an example rendezvous mission. The example shown is a rendezvous starting from an orbit with $r_1 = 8,000$ km and $i = 45°$ to a GEO orbit (all other orbital element values considered to be 0 at the initial time).

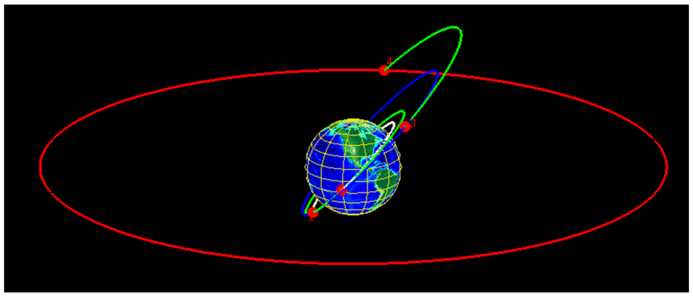
Figure 4.12: 4 Impulse Rendezvous

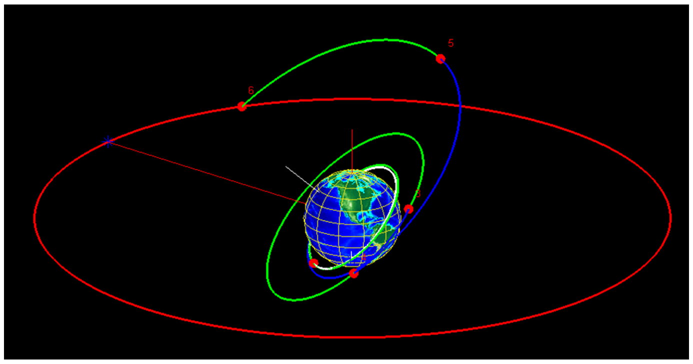
Figure 4.13: 6 Impulse Rendezvous

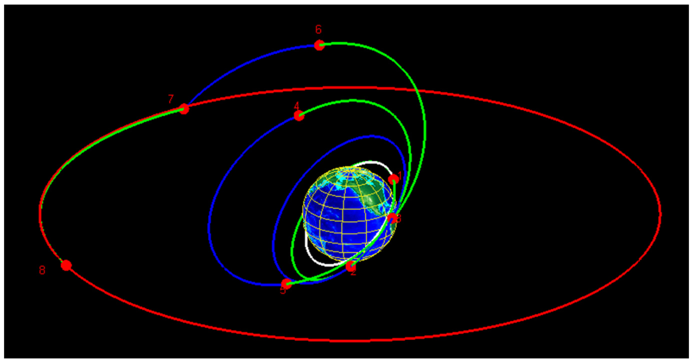
Figure 4.14: 8 Impulse Rendezvous

The plots show the progression of the impulses and exhibit a spiraling effect where a small amount of the inclination change is taken at each impulse until the final impulse where the majority of the inclination is removed. For a Monte Carlo test of 100 runs of the LA, the values obtained for the mean Δv value are shown in Table 4.8. Note that the values are all very similar, showing little (if any) gain by increasing the number of impulses.

Table 4.8: Orbit Rendezvous Mean Δv Values

Number of Impulses	Mean Δv Value
4	4.4438 km/s
5	4.4369 km/s
6	4.4501 km/s
7	4.4417 km/s
8	4.4302 km/s

Notice that the 6-impulse case exhibits a larger Δv value than the others, but that all cases are similar to within 20 m/s. Table 4.8 should not be interpreted as suggesting that more impulses provides a better solution (as this would unfairly imply that the optimal continuous thrust always provides the minimum Δv, which is known not to be the case). In the case of the 8-impulse rendezvous shown in Fig. 4.14, the 7^{th} impulse gets the spacecraft nearly into the correct orbit and requires just a very small (relative) final impulse to correct the phasing. This shows that the maneuver could be efficiently handled in fewer impulses.

A single case of the 4-impulse transfer is shown in Fig. 4.15. This behavior is more like a bi-elliptic transfer and produced a Δv value of 4.5198 km/s. Even though this value is larger, the algorithm did seek solutions that are outside of spiral trajectories only.

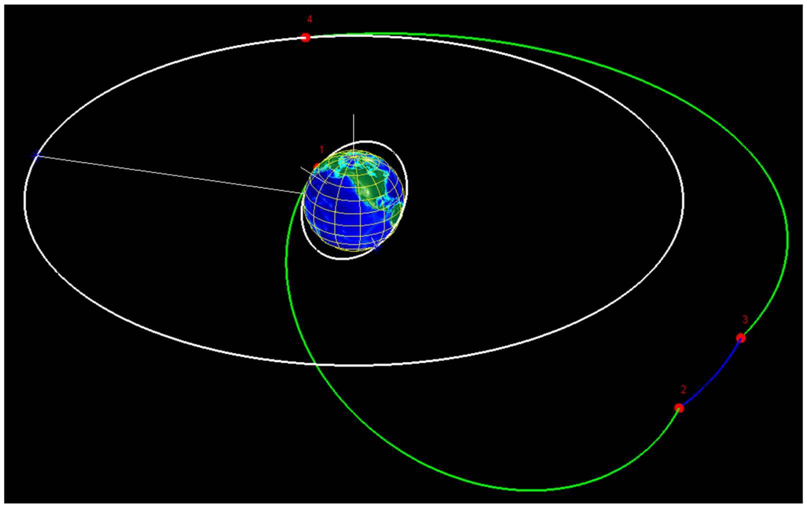

Figure 4.15: 4 Impulse Rendezvous

For the 4-impulse rendezvous example, a Monte Carlo test is performed and the statistics are shown for the GA and LA over 1000 tests in Table 4.9.

Table 4.9: Orbit Rendezvous Minimum Δv Values

Statistic	GA	LA
Mean Δv (km/s)	4.4682	4.4315
3σ of Δv (km/s)	0.1281	0.0401
Best Δv (km/s)	4.4391	4.4290
Worst Δv (km/s)	4.6912	4.4971

4.1.5 Final Numerical Local Minimization

As the Genetic Algorithm and Learning Approach are generally used as *global optimizers*, the addition of a numerical, *local minimization* technique was investigated. The built-in MATLAB function `fminunc` is used at the end of an optimization run with the best solution of the global minimization process being used as the starting point in the local minimization routine. This procedure met with limited success on all scenarios tested in this section. Generally, the improvement in the Δv value was at the cm/s level or smaller, and thus is not further reported. This verified that both the GA and LA functions performed well in finding a (at least locally) optimal solution.

4.1.6 Conclusions of N-Impulse Application

This section presented an algorithm for determining the minimum Δv in the N-impulse orbit transfer and rendezvous problems. The algorithm is based on novel constraints on the Δv selection which bound the choice of Δv such that the orbit after each impulse does not intersect the Earth (plus some altitude constraint) and a maximum, user-defined Δv value is not exceeded. The algorithm is shown to easily provide solutions for up to $N = 8$, which is significantly higher than previous examples, using the GA and the LA. The Learning Approach to Sampling Optimization and Genetic Algorithm were compared in terms of performance on several orbit transfer and rendezvous scenarios, including a Monte Carlo analysis of certain scenarios. It is shown that the Learning Approach consistently performed better by finding the lower Δv solution for all scenarios tested.

4.2 Periodic Close Encounters

The second application solved in this dissertation is the problem of Periodic Close Encounters in space. This work builds on the previous application in that the impulses derived in the previous section will be enforced to ensure that the problem is tractable.

4.2.1 Introduction

The problem of designing Periodic Close Encounters (PCE), where one spacecraft periodically encounters the orbit of a second spacecraft, is first proposed by Clocchiatti and Mortari[90]. The initial idea is then expanded and validated in Clocchiatti's MS thesis[91] and finally reached it's current version which is reported in this dissertation. The 2-impulse problem is defined by the geometry shown in Fig. 4.16

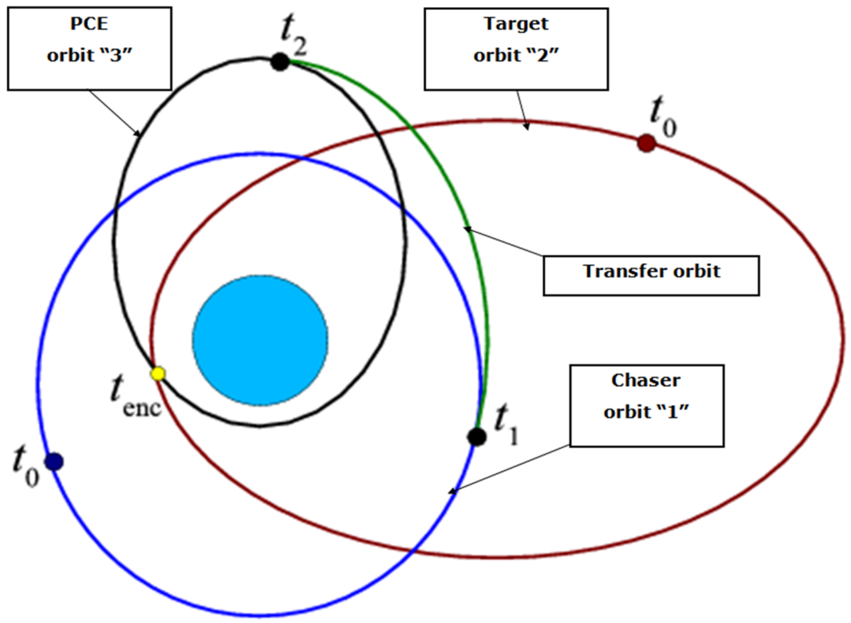

Figure 4.16: 2-impulse PCE Geometry and Definitions

The PCE problem is what can be called an *open rendezvous problem*, meaning two satellites arrive very close to each other (with sufficient minimum distance to avoid an actual collision) in space at the same time. However, the velocity components are not necessarily equivalent, as this would be a rendezvous.

From the 2-impulse maneuver geometry shown in Fig. 4.16, the PCE problem is to find an orbit "**3**" for a satellite (chaser), initially on orbit "**1**," which must periodically encounter a second satellite (target), on orbit "**2**," at a to-be-defined location and time. The goal of the chaser is to arrive in the final orbit "**3**", called the PCE orbit, which meets user-specified requirements, in a fuel-optimal transfer. The indices **1**, **2**, and **3**, is used as subscript to identify the chase, target, and PCE orbits, respectively. However, one purpose of using this application in this dissertation is to extend the work by Clocchiatti and Mortari into N-impulses. As such, the subscript **3** will denote the final PCE orbit with the understanding that there may be intermediate transfer orbits. The notation is kept the same as previous literature for convenience.

The following conditions must be satisfied by the PCE orbit:

1. be sufficiently close (in terms of Δv_{tot}) to orbit "**1**",
2. be compatible (or *resonant*) with orbit "**2**", and
3. encounter the target spacecraft with prescribed distance and observation time requirements.

Solving the PCE problem is motivated from both civilian and military space asset needs. A civilian need can be to inspect an asset (or set of assets) to better design an expensive repair mission. From a military point of view, the capability of performing PCE in space can be used for intelligence and can be seen as an affordable and immediate answer to respond to threats to international space assets.

The original PCE theory[90, 91], consists of only a 2-impulse orbit transfer strategy to allow responsive close-encounters for space surveillance. This technique clearly provided orbit transfer costs lower than a rendezvous transfer in terms of Δv_{tot} used in the maneuver, while complying with assigned minimum observation time/day. In Refs. [90, 91] a Lambert solver[48] is proposed to find the two impulses, the integers involved in the compatible PCE orbit are selected using `bitstring`, and the optimization problem is solved using GAs. Unfortunately, the resulting optimization problem demonstrated slow convergence and, in many cases, converged to some expensive, undesirable, local minima. It failed because too many chromosomes are generating Earth-impacting or hyperbolic trajectories due to the method devised to generate potential solutions.

The PCE problem is a highly non-linear, multi-minima problem requiring a compatible orbit. Mathematically, this requires that the variables include both discrete (i.e., integer) and real design parameters, $\mathbb{Z}$ and $\Re$, respectively. For this reason, traditional, analytical optimization tools cannot be adopted and numerical techniques, such as evolutionary algorithms, appear to be a proper solution tool. In particular, GAs are selected to solve orbital mechanics problems like spacecraft rendezvous[85], Earth surveillance problems[92, 93], and trajectory optimization problems[82, 83]. For the problems solved in Refs. [85, 86, 92, 93], as well as for the PCE problem, the computational load of the fitness function is strongly reduced by minimizing orbit propagations and by solving the final orbit transfer using Lambert's solver[48].

Using the previously learned lessons, the initial approach to solve PCE problem[90, 91] is improved on in an AFRL-sponsored study[87] where two important improvements were obtained. The first improvement is making better use of the GA optimization; that is, using it such that *no chromosome yields an unfeasible solution.* This led to an analytical study and derivation of generating admissible impulses by finding bounds on velocity changes to avoid Earth-impacting and highly eccentric trajectories as presented in the previous section.

The PCE problem only makes sense when applied to real-world spacecraft. Thus, operational considerations must be given in the form of constraints. An example of constraints defined in the PCE problem is shown in Table 4.10 below. These constraints are adopted in the current version along with some new constraints, as described below.

Table 4.10: Example PCE Constraints

Parameter		Value
Minimum perigee altitude	=	350 km
Minimum observation time	=	60 s/day
Maximum repetition time	=	2 day
Maximum encounter time	=	1 day
Maximum Δv per maneuver	=	5 km/s
Maximum observation distance	=	10 km
Maximum allowed eccentricity	=	0.9

4.2.2 PCE Features and Constraints

The adoption of the analytical velocity bounds greatly reduced the search space and allowed a proper use of the optimization technique. This update allowed the extension of the previous 2-impulse version to the N-impulse technique to be described here. A detailed description of admissible impulses to the N-impulse orbit transfer and rendezvous problem is given in Ref. [88] and additionally in the previous section.

In addition to the velocity bounds, the current PCE program contains the following new features:

1. minimum encounter distance,
2. minimum dwell time,
3. user-defined time constraints,
4. illumination requirements are used to pre-process allowable encounter times,
5. GA replaced by Learning Approach (LA) optimization technique.

The following subsections describe the above new features and how they are implemented. The general flow of the algorithm as well as all the mathematical proofs and derivations are provided in Ref. [87].

4.2.2.1 Minimum Encounter Distance

The minimum encounter distance (e.g., $d_{\min} = 10$ m) is enforced by changing the value of the radius at encounter toward the Sun direction if illumination constraints are defined. In other words, the encounter radius for the chaser is set

$$\mathbf{r}_{3e} = \mathbf{r}_{2e} \pm d_{\min}\,\hat{\mathbf{r}}_e \tag{4.36}$$

where the sign is chosen to maximize illumination of the target relative to the chaser and $\hat{\mathbf{r}}_e$ is the unit vector in the direction of the radius at encounter.

4.2.2.2 Minimum Dwell Time

The first encounter time is a variable that is optimized during the algorithm. Once the first encounter time is chosen, the position and velocity of the target orbit, $\mathbf{r}_e$ and $\mathbf{v}_{2e}$, are determined by orbit propagation. The energy equation then allows the velocity of the chaser at the first encounter time to be computed as

$$v_{3e} = \sqrt{\frac{2\mu}{r_e} - \frac{\mu}{a_3}} \tag{4.37}$$

where a_3 is the semi-major axis of the PCE orbit and is known from the compatibility condition, $k_2 T_2 = k_2 T_3$, where T_2 and $T_3 = 2\pi\sqrt{a_3^3/\mu}$ are the orbital periods of the target and PCE orbits, respectively, and k_2 and k_3 are two integers.

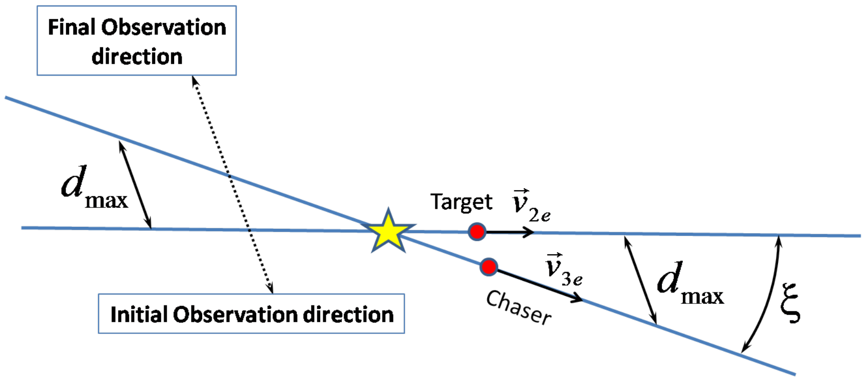

Figure 4.17: Encounter Geometry

Figure 4.17 shows the relevant geometry. The velocity triangle allows the following equation

$$v_{\text{rel}}^2 = v_{2e}^2 + v_{3e}^2 - 2\mathbf{v}_{2e}^{\mathrm{T}}\mathbf{v}_{3e} \tag{4.38}$$

Assuming that the observation time is much smaller than the orbital period allows the relative distance equation to be linearized such that $d_{\text{rel}} = |v_{\text{rel}}|\,t$. Therefore, we have

$$d_{\max} = |v_{\text{rel}}|\frac{\Delta t_{\text{o}}}{2} \tag{4.39}$$

and solved for the time gives

$$\Delta t_{\text{d}} = \Delta t_{\text{o}}\frac{86400}{T_3} \geq \Delta t_{\min\text{ d}} \tag{4.40}$$

where the notation Δt_o is the dwell time per orbit, Δt_d is the dwell time per day, and $\Delta t_{\text{min d}}$ is the minimum required dwell time per day. This leads to

$$\cos \xi_{\max} = \hat{\mathbf{v}}_{2e}^{\mathrm{T}} \hat{\mathbf{v}}_{3e} = \frac{v_{2e}^2 + v_{3e}^2 - [2 \cdot 86400\, d_{\max}/(T_3\, \Delta t_{\text{min d}})]^2}{2\, v_{2e}\, v_{3e}} \tag{4.41}$$

Thus, by specifying a maximum angle, $\xi_{\max}$, between the target and chaser velocities at the encounter, the dwell time requirement is enforced. This requirement is only met when the minimum dwell time is small when compared to the orbital period.

There are three cases to investigate:

1. $v_{rel} < |v_{2e} - v_{3e}| \quad \rightarrow \quad \Delta t_d < \Delta t_{\text{min d}}$: unfeasible solution
2. $|v_{2e} - v_{3e}| \leq v_{rel} \leq v_{2e} + v_{3e} \quad \rightarrow \quad \Delta t_d \geq \Delta t_{\text{min d}}$: solutions feasible for $\xi \leq \xi_{\max}$
3. $v_{rel} > v_{2e} + v_{3e} \quad \rightarrow \quad \Delta t_d \geq \Delta t_{\text{min d}}$: always feasible solutions

Equation (4.37) provides us the magnitude of the PCE orbit's velocity at the encounter time. However, its direction is unknown. To satisfy the observation requirements, the direction of $\hat{v}_{3e}$ must be inside a cone of axis $\hat{v}_{2e}$ and aperture $\xi_{\max}$ as provided by Eq. (4.41). This is done in two steps. First, we define the direction of $\hat{v}_{3e}^*$ that lie on the plane defined by $[\hat{r}_e, \hat{v}_{2e}]$. This direction is

$$\hat{v}_{3e}^* = \hat{r}_e \sin \xi + \hat{v}_{2e} \sin(\lambda - \xi) \qquad \text{where} \qquad \cos \lambda = \hat{r}_e^{\mathrm{T}} \hat{v}_{2e} \tag{4.42}$$

and then the velocity vector, $\mathbf{v}_{3e}$ is obtained by a rigid rotation about $\hat{v}_{2e}$ by a to-be-determined angle, $\alpha \in [0, 2\pi)$

$$\mathbf{v}_{3e} = v_{3e} R\left(\hat{v}_{2e}, \alpha\right) \hat{v}_{3e}^* \tag{4.43}$$

Figure 4.18 illustrates the relationships of the angles.

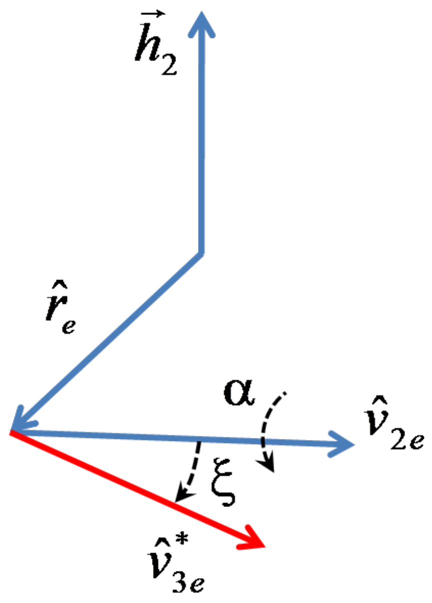

Figure 4.18: Geometry of ξ and α to Define the $\hat{v}_{3e}$ Direction

4.2.2.3 Time Constraints

Additional constraints depending on time only, such as illumination requirements during the observation or a requirement of performing the observation over some specified regions (for real-time observations such as observing the target spacecraft from both space and Earth), are implemented by introducing a normalized fictitious continuous time, δ. The normalized time is generated by expressing the constrained time by cutting out the time ranges that are not considered. Time constraints are particularly important to perform surveillance of repeating ground-track target satellites.

This idea is depicted in Fig. 4.19, where the continuous times, δ_k, are defined

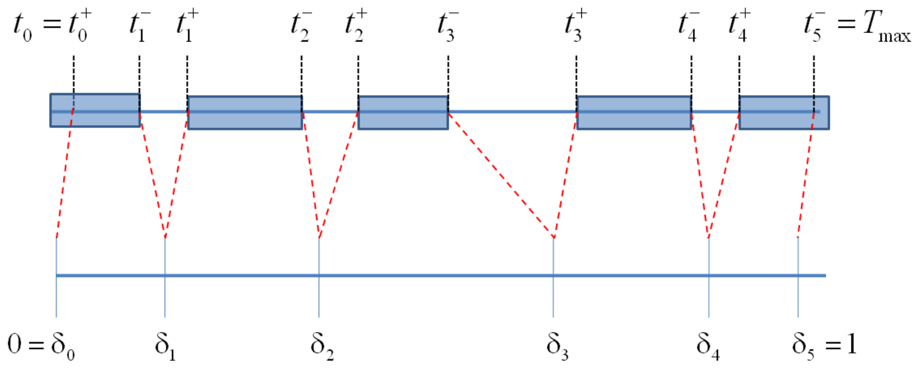

Figure 4.19: Admissible Time Ranges and Continuous Normalized Time

according to the following sequence

$$\delta_k = \frac{t_k^- - t_{k-1}^+}{T_\delta} + \delta_{k-1} \qquad \text{where} \qquad T_\delta = \sum_{i=1}^{n} t_i^- - t_{i-1}^+ \tag{4.44}$$

Therefore, we have $\delta_0 = 0$ for $t = t_0$ and $\delta_N = 1$ for $t = T_{\max}$.

One possible realization of this capability is to restrict the encounter time to times when the target satellite is not eclipsed by the Earth (e.g., observation in the visible spectrum).

The current PCE program also allows to specify the encounter time. This is a useful option (see third example, Scenario I of the "Numerical Examples" section) when the encounter has to occur over an assigned Earth region (e.g., for real-time observations *and* communications). This feature is simply implemented by normalizing all of the $(N + 1)$ times by setting the last one, t_{N+1}, equal to the assigned encounter time t_e.

4.2.2.4 Illumination Requirements

Besides being able to specify particular windows of opportunity that provide good conditions for observation, the cost function is augmented (by using a pre-processor) to allow for illumination to play a role in the optimization process. Consider a satellite with a (passive) camera as the observation instrument with the Sun as the illumination source. The question is, what constitutes favorable illumination conditions? A first requirement is that the target satellite not be in eclipse at the encounter time. But that is only a necessary (not the optimal) condition for sufficient illumination.

To obtain the optimal condition, we first define a constraint angle indicating the cone of illumination. Figure 4.20 shows the geometry of the illumination angle constraint. This involves the velocity $\mathbf{v}_{2e}$, the plane orthogonal to $\mathbf{v}_{2e}$, and the direction of the Sun, $\mathbf{r}_{sun} - \mathbf{r}_e$. The user defines a maximum value of the angle $\bar{\beta}$, where the angle between the Sun's rays and $\mathbf{v}_{2e}$ directions is $\dfrac{\pi}{2} - \bar{\beta}$. This means that if $\bar{\beta} = 0$, the Sun's rays are orthogonal to $\mathbf{v}_{2e}$. The angle $\bar{\beta}$ defined in Fig. 4.20

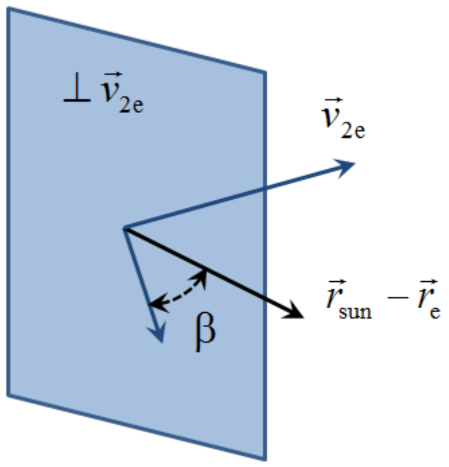

Figure 4.20: Illumination Angle Constraint

is an estimated value of the true angle β that is defined as the angle between the Sun's rays direction and the direction orthogonal to the plane where $\mathbf{v}_{2e}$ and $\mathbf{v}_{3e}$ lie.

The actual value of β is computed using the PCE orbit velocity which cannot be pre-computed. However, the angluar value is constrained to be within a user-defined cone.

As an example, consider an illumination angle constraint of $\beta_{\max} = 30°$. Figure 4.21 shows, for a typical LEO orbit (7000 km semi-major axis, uninclined and circular), the times when the illumination angle constraint is satisfied. By knowing the times, the position and velocity vectors can be quickly computed.

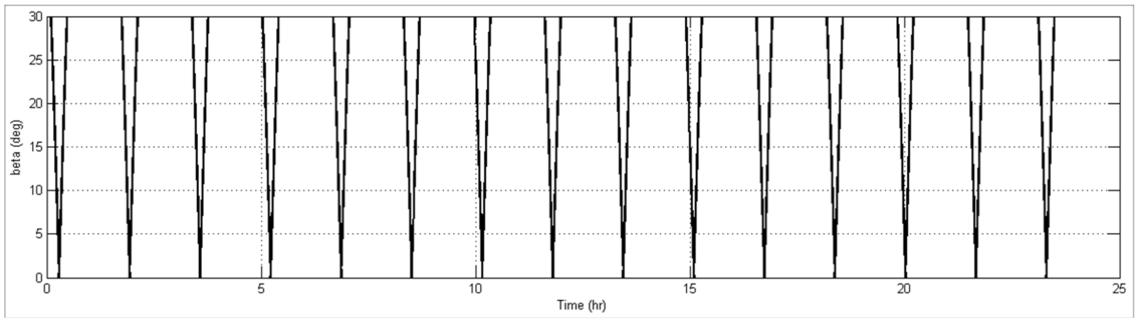

Figure 4.21: Illumination Angle Constraint, $\beta = 30\,\mathrm{deg}$

Observing that the chaser satellite is pointing almost entirely within the plane defined by the two satellites' velocity vectors throughout the encounter (i.e., a linear assumption), we define optimal illumination as having the vector to the Sun perpendicular to that plane. Having the Sun lie in that plane guarantees that the chaser is looking directly into the Sun as much as half of the observation, while having the Sun perpendicular to it guarantees that at least half the target is fully lit when observed by the chaser without ever being backlit by the Sun. Figure 4.22 illustrates the angle of interest, β.

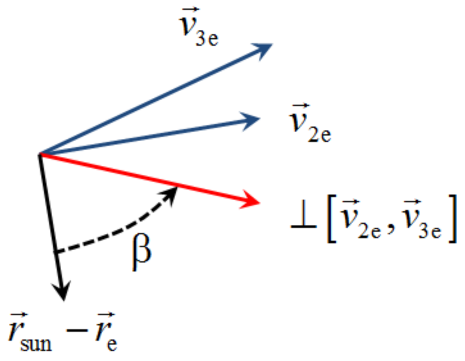
$\vec{v}_{3e}$
$\vec{v}_{2e}$
$\perp[\vec{v}_{2e}, \vec{v}_{3e}]$
β
$\vec{r}_{sun} - \vec{r}_{e}$

Figure 4.22: Illumination Angle β at Encounter

From Fig. 4.22 it is seen that the encounter is illuminated by the Sun if

$$\beta = \cos^{-1}\left(\frac{\mathbf{v}_{2e} \times \mathbf{v}_{3e}}{|\mathbf{v}_{2e} \times \mathbf{v}_{3e}|} \cdot \frac{\mathbf{r}_{\text{sun}} - \mathbf{r}_{e}}{|\mathbf{r}_{\text{sun}} - \mathbf{r}_{e}|}\right) \leq \beta_{\max} \tag{4.45}$$

The illumination constraints are pre-processed since the times when the target is illuminated do not change, as the target orbit is fixed. Thus, the times when the target is illuminated is first computed and those times are constrained by the method shown in Section 2c.

4.2.3 PCE Algorithm

The N-impulse PCE problem has a search space dimensionality (i.e., number of independent variables) of $(4N - 2)$. These variables are defined as follows:

1. one variable (or index) identifies the pair of integers, $[k_2, k_3]$, associated with the PCE compatibility constants as defined by

$$k_2\, T_2 = k_3\, T_3 = T_{\text{rep}} \tag{4.46}$$

where T_{rep} is the repetition time of the encounter;

2. a set of $(N+1)$ consecutive times defining the impulses, t_k, and encounter, $t_e \leq t_{e\,\max}$, times such that

$$t_0 \;\leq\; t_1 \overset{\text{T1}}{\rightarrow} t_2 \overset{\text{T2}}{\rightarrow} t_3 \;\cdots\; t_{N-2} \overset{\text{T}(N-2)}{\rightarrow} t_{N-1} \overset{\text{T}(N-1)}{\rightarrow} t_n \overset{\text{PCE}}{\rightarrow} t_e \;\leq\; t_{e\,\max}$$

   where t_e is the first encounter time and $t_{e\,\max}$ is its maximum allowed value;

3. one angle, ξ, defining the angle between the chaser and target velocities at the encounter (identifying the dwell time) as shown in Fig. 4.18;

4. one angle, α, identifying how the vector $\mathbf{v}_{3e}$ is rotated about $\mathbf{v}_{2e}$ (to remain within the cone of aperture $\xi_{\max}$) as shown in Eq. (4.41); and

5. $(3N-6)$ variables to define the first $(N-2)$ admissible impulse components[1]

To initialize the PCE algorithm, the number of allowed impulses and the constraints shown in Table 4.10 must be specified. The constraints placed on illumination (if any) are pre-computed as the orbit of the target is assumed to be constant (i.e., no maneuvering takes place) based on the illumination cone defined by the user. Once the values are set and the illumination computed, the cost function is used with the LA numerical optimization technique. The cost function value that is minimized is the total $J = \displaystyle\sum_{k=1}^{N} |\, \Delta v_k \,|$ of the maneuver from the chaser orbit to the PCE orbit (using up to N-impulses). The flow of the cost function is shown in Algorithm 13. The LA is the optimization algorithm that finds the combination of variables to provide the minimal Δv value.

[1]The last two impulse components are provided by the Lambert solver applied for the last transfer orbit whose time-of-flight is $(t_N - t_{N-1})$. See Ref. [88] for more details.

Algorithm 13 Cost Function Pseudo-code

```
1: Select integer index, $k \in [1, k_{\max}]$, which defines $a_3$, the semi-major axis of the PCE orbit using Eq. (4.52)
2: Select impulse and encounter times sequentially, $t_1, t_2, \ldots, t_N, t_e$
3: Propagate the target orbit from $t_0$ to $t_e$
4: Compute $\mathbf{r}_e$, the position at the encounter
5: Compute the velocity magnitude of the PCE orbit at encounter, $v_{3e}$, using Eq. (4.37)
6: Select $\xi \in [0, \xi_{\max}]$, as defined in Eq. (4.41) and $\alpha \in [0, 2\pi)$, which define the velocity vector at encounter, $\mathbf{v}_{3e}$
7: if $\mathbf{v}_{3e}$ causes perigee to be too low then
8:     $J = \infty$
9: else
10:    for $i = 1$ to $(N - 2)$ do
11:        Propagate to $t_i$ and determine $\mathbf{r}_i$ and $\mathbf{v}_i$
12:        Apply admissible impulse $\Delta v_i$ component values from Eqs. (4.20) and Eq. (4.28)
13:    end for
14:    Propagate to $t_{N-1}$ and determine $\mathbf{r}_{N-1}$ and $\mathbf{v}_{N-1}$
15:    Compute $\Delta v_F$ by solving Lambert's problem using $\mathbf{r}_{N-1}$, $\mathbf{r}_N$, and $(t_N - t_{N-1})$
16:    $J = \Delta v_F + \sum_{i=1}^{N-2} \Delta v_i$
17: end if
```

4.2.3.1 Compatible Orbit Selection

Consider the maximum time interval between two consecutive encounters, $T_{\max}$, as assigned by the user. The compatibility condition is bounded by integer multiples of the respective orbit periods as

$$T_2 \leq k_2 T_2 = k_3 T_3 \leq T_{\max} \tag{4.47}$$

where k_2 is constrained to the range

$$1 \leq k_2 \leq k_{2\max} = \left\lfloor \frac{T_{\max}}{T_2} \right\rfloor \tag{4.48}$$

The value of k_3 is similarly constrained to the range

$$1 \leq k_3 \leq k_{3\max} = \left\lfloor \frac{T_{\max}}{T_{3\min}} \right\rfloor \tag{4.49}$$

Thus, all rational values of k_2/k_3 are chosen as the first variable in the cost function (Algorithm 13) from the range

$$\frac{T_{3\min}}{T_2} \leq \frac{k_2}{k_3} \leq \frac{T_{3\max}}{T_2} \tag{4.50}$$

The normalized times, δ_i, are then generated sequentially as

$$\delta_i = \frac{1}{N+1} \sum_{j=1}^{i} x_j \qquad i = 0, 1, \ldots, N \tag{4.51}$$

where x_j is a random number in $[0, 1]$ and the times have been normalized. From δ_i, we can compute the actual time, t_i.

The orbital period (and therefore semi-major axis) of the PCE orbit is derived from

$$T_3 = \frac{k_2}{k_3} T_2 \qquad \rightarrow \qquad a_3 = \sqrt[3]{\mu [T_3/(2\pi)]^2} \tag{4.52}$$

The semi-major axis allows a maximum apogee radius of

$$R_{3a\max} = 2a_3 - R_{p\min} \leq R_{2e}$$

where R_{2e} is the modulus of the radius at the encounter. By using the first encounter time, $t_e \in [t_0, t_0 + T_{\max}]$, the target orbit is propagated up to t_e. This allows the evaluation of $\mathbf{r}_e$ and $\mathbf{v}_{2e}$. The semi-major axis is defined by the compatibility index as shown in Eq. (4.52). Therefore, the energy equation, Eq. (4.37), is used to derive the velocity at the encounter. Once the modulus of the velocity at the encounter time is known, as evaluated in Eq. (4.37), the direction of the velocity at the encounter is determined using the two variables, ξ and α.

The purpose of the LA is to find the combination of variables that minimizes the value of the cost function, J, as shown in Step 16 of Algorithm 13. The next section will show numerical examples using real satellites.

4.2.4 Numerical Examples

This section shows various examples of (impulsive) PCE maneuvers. In order to highlight the various possibilities offered by this technique, we have selected a common chaser, the ARIANE-44L (SatID: 28576), a satellite in an elliptical, equatorial orbit, whose Two Line Elements (TLE) are

```
ARIANE 44L
1 28576U 91075N   08351.94568414  .00000179  00000-0  64019-2 0  6927
2 28576 006.5534 128.0629 6595687 237.3611 042.0029 02.83587463 72170
```

and two different target spacecraft

1. ALSAT-1 (SatID: 27559), a satellite in a polar orbit, whose TLE are

```
ALSAT 1
1 27559U 02054A   08259.52685948 -.00000002  00000-0  84653-5 0  6025
2 27559 097.9807 137.4784 0009664 216.5494 143.5047 14.62977897309534
```

2. COSMOS-2350 (SatID 25315), a satellite in GEO orbit, whose TLE are

```
COSMOS 2350
1 25315U 98025A   08352.73192245 -.00000078  00000-0  10000-3 0  2950
2 25315 006.9388 064.3948 0002775 320.1585 039.6957 01.00258154 38973
```

In all of the numerical examples given in this section, the same values of the PCE constraints have been used. These constraints are given in Table 4.10.

4.2.4.1 Example #1: ARIANE-44L to ALSAT-1

Three different scenarios have been selected for the PCE mission using ARIANE-44L as the chaser and ALSAT-1 as the target. The main results of these scenarios are summarized in Table 4.11 and described in detail below. The initial time for these scenarios was 16-Dec-2008 at 22:41:47 (UTC) as this is when the TLEs were available.

Table 4.11: ARIANE-44L to ALSAT-1 Example, Results Summary

Scenario	Number of Impulses	Max Illumination angle (deg)	Δv_{tot} (km/s)	Illumination angle (deg)
I	3	90.0	4.2684	58.2441
II	3	30.0	4.6041	2.1591
III	2	90.0	4.9319	53.2842

4.2.4.2 Scenario I

Scenario I is 3 impulses with $\beta = 90\,\text{deg}$, meaning the illumination angle is not involved in the cost function. The LA determined that the first encounter time (UTC) is 17-Dec-2008 at 15:18:31. The value determined is $\Delta v = 4.2684$ km/s. Table 4.12 shows the impulses and Fig. 4.23 shows a 3D depiction of the orbits, including the transfer trajectory.

Table 4.12: ARIANE-44L to ALSAT-1 Example, Scenario I

$\|\Delta v_1\| = 0.6152$ km/s	Time (UTC): 17-Dec-2008, 04:25:41		
Position (km)	$R_x = -19353.6$	$R_y = -16744.4$	$R_z = 2936.5$
Velocity (km/s)	$V_x = 3.4090$	$V_y = -0.5876$	$V_z = -0.2669$
Δv (km/s)	$\Delta v_r = -0.2735$	$\Delta v_t = 0.5263$	$\Delta v_n = 0.1634$

$\|\Delta v_2\| = 2.8945$ km/s	Time (UTC): 17-Dec-2008, 09:42:05		
Position (km)	$R_x = -28312.6$	$R_y = 25467.4$	$R_z = -1181.4$
Velocity (km/s)	$V_x = -2.4198$	$V_y = -0.7178$	$V_z = 0.2163$
Δv (km/s)	$\Delta v_r = -0.8487$	$\Delta v_t = -2.0612$	$\Delta v_n = 1.8465$

$\|\Delta v_3\| = 0.7587$ km/s	Time (UTC): 17-Dec-2008, 14:58:41		
Position (km)	$R_x = 4267.8$	$R_y = -3219.1$	$R_z = 7644.7$
Velocity (km/s)	$V_x = 2.9009$	$V_y = -3.2035$	$V_z = -7.0352$
Δv (km/s)	$\Delta v_r = 0.4054$	$\Delta v_t = -0.6031$	$\Delta v_n = -0.2181$

4.2.4.3 Scenario II

Scenario II is 3 impulses with $\beta = 30\,\text{deg}$, meaning the illumination angle is constrained. The LA determined that the first encounter time (UTC) is 17-Dec-2008 at 13:45:43. The value determined is $\Delta v = 4.6041$ km/s. The encounter illumination angle computed by the LA is 0.96804 deg. Table 4.13 shows the impulses and Fig.

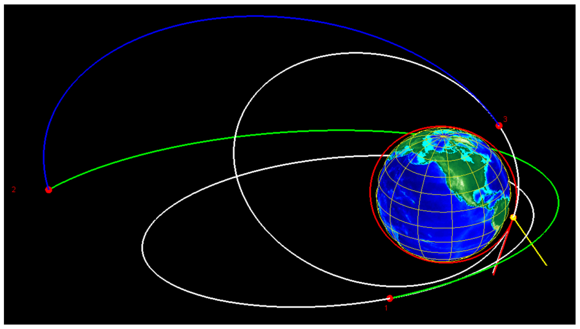

Figure 4.23: 3D view of ARIANE-44L to ALSAT-1 PCE mission (Scenario I)

4.24 shows a 3D depiction of the orbits, including the transfer trajectory.

Table 4.13: ARIANE-44L to ALSAT-1 Example, Scenario II

$\lvert\Delta v_1\rvert = 0.5095$ km/s	Time (UTC): 17-Dec-2008, 04:27:38		
Position (km)	$R_x = -18950.9$	$R_y = -16810.6$	$R_z = 2904.8$
Velocity (km/s)	$V_x = 3.4622$	$V_y = -0.5411$	$V_z = -0.2750$
Δv (km/s)	$\Delta v_r = -0.4775$	$\Delta v_t = 0.1686$	$\Delta v_n = 0.05597$

$\lvert\Delta v_2\rvert = 3.4761$ km/s	Time (UTC): 17-Dec-2008, 08:06:07		
Position (km)	$R_x = -22088.7$	$R_y = 20011.7$	$R_z = 0.5$
Velocity (km/s)	$V_x = -3.3153$	$V_y = -0.2893$	$V_z = 0.2815$
Δv (km/s)	$\Delta v_r = -0.9267$	$\Delta v_t = -2.4050$	$\Delta v_n = 2.3325$

$\lvert\Delta v_3\rvert = 0.6185$ km/s	Time (UTC): 17-Dec-2008, 13:22:48		
Position (km)	$R_x = 5506.3267$	$R_y = -4142.2545$	$R_z = 6742.9732$
Velocity (km/s)	$V_x = 1.5288$	$V_y = -2.3197$	$V_z = -7.4467$
Δv (km/s)	$\Delta v_r = 0.3253$	$\Delta v_t = -0.4745$	$\Delta v_n = -0.2271$

4.2.4.4 Scenario III

Scenario III is 2 impulses with $\beta = 90$ deg, meaning the illumination angle is not used in the cost function. The LA determined that the first encounter time (UTC)

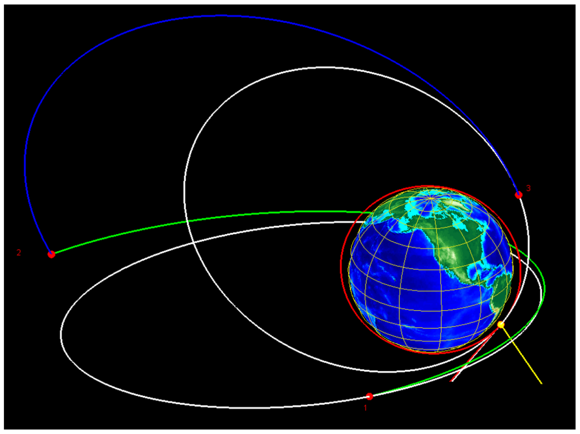

Figure 4.24: 3D view of ARIANE-44L to ALSAT-1 PCE mission (Scenario II)

is 17-Dec-2008 at 03:55:48. The value determined is $\Delta v = 4.9319$ km/s. Table 4.14 shows the impulses and Fig. 4.25 shows a 3D depiction of the orbits, including the transfer trajectory.

Table 4.14: ARIANE-44L to ALSAT-1 Example, Scenario III

$\|\Delta v_1\| = 4.4290$ km/s	Time (UTC): 16-Dec-2008, 23:02:50		
Position (km)	$R_x = -16999.4$	$R_y = 14260.4$	$R_z = 528.5$
Velocity (km/s)	$V_x = -4.0558$	$V_y = -0.6246$	$V_z = 0.4112$
Δv (km/s)	$\Delta v_r = -0.7696$	$\Delta v_t = -3.1099$	$\Delta v_n = 3.0581$

$\|\Delta v_2\| = 0.5029$ km/s	Time (UTC): 17-Dec-2008, 03:30:15		
Position (km)	$R_x = 6393.0$	$R_y = -4323.4$	$R_z = 6797.5$
Velocity (km/s)	$V_x = 0.9354$	$V_y = -1.8361$	$V_z = -7.1055$
Δv (km/s)	$\Delta v_r = 0.2648$	$\Delta v_t = -0.2801$	$\Delta v_n = -0.3230$

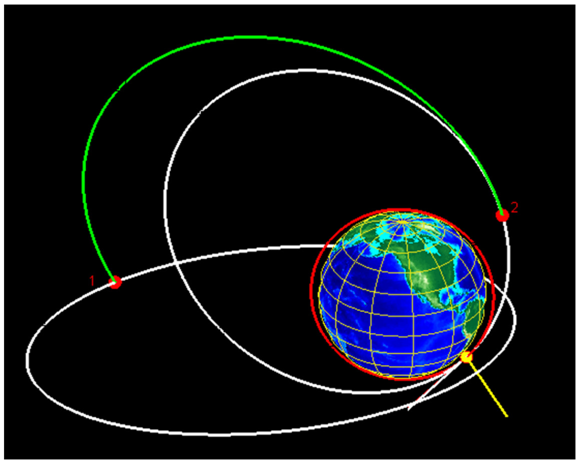

Figure 4.25: 3D view of ARIANE-44L to ALSAT-1 PCE mission (Scenario III)

4.2.4.5 Example #2: ARIANE-44L to COSMOS-2350

Three different scenarios are selected for the PCE mission using ARIANE-44L as chaser and COSMOS-2350 (ID=25315) as target. The main results of these scenarios are summarized in Table 4.15 and detailed below. The initial time for these scenarios was 17-Dec-2008 at 17:33:58 (UTC) as this is when the TLEs were available.

Table 4.15: ARIANE-44L to COSMOS-2350 Example, Results Summary

Scenario	Number of Impulses	Max Illumination angle (deg)	Δv_{tot} (km/s)	Illumination angle (deg)
I	3	0.0	1.7270	69.1054
II	2	0.0	1.7917	69.4127
III	2	10.0	2.3573	0.86442

4.2.4.6 Scenario I

Scenario I is 3 impulses with $\beta = 90\,\text{deg}$, meaning the illumination angle is not involved in the cost function. The LA determined that the first encounter time

Table 4.16: ARIANE-44L to COSMOS-2350 PCE, Scenario I

$\lvert\Delta v_1\rvert = 0.0039$ km/s	Time (UTC): 17-Dec-2008, 22:00:36		
Position (km)	$R_x = -9877.2$	$R_y = -16694.0$	$R_z = 2075.5$
Velocity (km/s)	$V_x = 4.5883$	$V_y = 0.8243$	$V_z = -0.4735$
Δv (km/s)	$\Delta v_r = 0.0015$	$\Delta v_t = 0.0035$	$\Delta v_n = 0.0009$

$\lvert\Delta v_2\rvert = 1.3706$ km/s	Time (UTC): 18-Dec-2008, 02:46:52		
Position (km)	$R_x = -34441.8$	$R_y = 728.0$	$R_z = 3057.0$
Velocity (km/s)	$V_x = -0.4508$	$V_y = -1.9800$	$V_z = 0.1811$
Δv (km/s)	$\Delta v_r = -0.0334$	$\Delta v_t = 1.3450$	$\Delta v_n = -0.2616$

$\lvert\Delta v_3\rvert = 0.3525$ km/s	Time (UTC): 18-Dec-2008, 07:39:34		
Position (km)	$R_x = -3878.0$	$R_y = -37846.4$	$R_z = 52.7$
Velocity (km/s)	$V_x = 3.0232$	$V_y = -0.2503$	$V_z = -0.2698$
Δv (km/s)	$\Delta v_r = 0.0331$	$\Delta v_t = 0.3459$	$\Delta v_n = -0.0595$

(UTC) is 18-Dec-2008 at 13:31:54. The value determined is $\Delta v = 1.7270$ km/s. Table 4.16 shows the impulses and Fig. 4.26 shows a 3D depiction of the orbits, including the transfer trajectory.

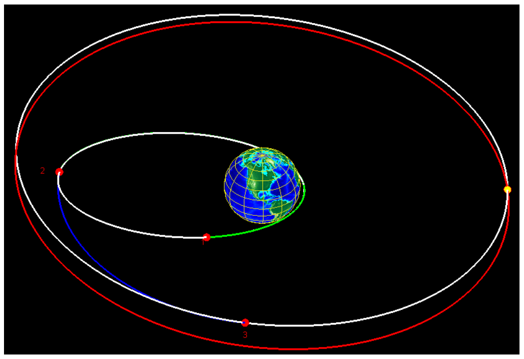

Figure 4.26: 3D view of ARIANE-44L to COSMOS-2350 PCE mission (Scenario I)

Table 4.17: ARIANE-44L to COSMOS-2350 PCE, Scenario II

$\lvert\Delta v_1\rvert = 0.9267$ km/s	Time (UTC): 18-Dec-2008, 01:33:43		
Position (km)	$R_x = -29127.9$	$R_y = 8948.2$	$R_z = 2002.0$
Velocity (km/s)	$V_x = -2.0303$	$V_y = -1.7264$	$V_z = 0.3059$
Δv (km/s)	$\Delta v_r = -0.1256$	$\Delta v_t = 0.8937$	$\Delta v_n = -0.2106$

$\lvert\Delta v_2\rvert = 0.86502$ km/s	Time (UTC): 18-Dec-2008, 05:18:07		
Position (km)	$R_x = -29356.8$	$R_y = -25106.8$	$R_z = 2242.9$
Velocity (km/s)	$V_x = 1.6986$	$V_y = -1.8231$	$V_z = -0.1082$
Δv (km/s)	$\Delta v_r = -0.1188$	$\Delta v_t = 0.8337$	$\Delta v_n = -0.1977$

4.2.4.7 Scenario II

Scenario II is 2 impulses with $\beta = 90\,\text{deg}$, meaning the illumination angle is not involved in the cost function. The LA determined that the first encounter time (UTC) is 18-Dec-2008 at 13:15:30. The value determined is $\Delta v = 1.7917$ km/s. Table 4.17 shows the impulses and Fig. 4.27 shows a 3D depiction of the orbits, including the transfer trajectory.

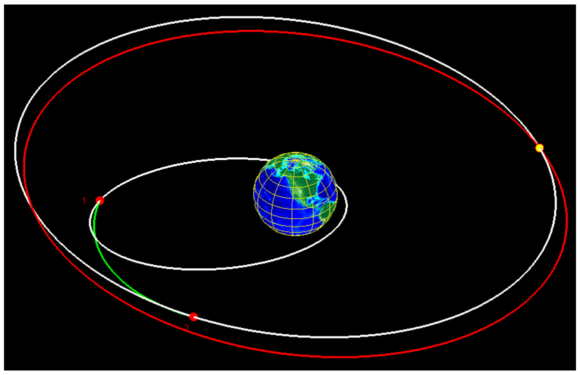

Figure 4.27: 3D view of ARIANE-44L to COSMOS-2350 PCE mission (Scenario II)

Table 4.18: ARIANE-44L to COSMOS-2350 PCE, Scenario III

$\lvert\Delta v_1\rvert = 1.9777$ km/s	Time (UTC): 18-Dec-2008, 01:25:20		
Position (km)	$R_x = -28054.2$	$R_y = 9800.0$	$R_z = 1844.5$
Velocity (km/s)	$V_x = -2.2394$	$V_y = -1.6578$	$V_z = 0.3200$
Δv (km/s)	$\Delta v_r = -1.0810$	$\Delta v_t = 1.5763$	$\Delta v_n = -0.5078$

$\lvert\Delta v_2\rvert = 0.3796$ km/s	Time (UTC): 18-Dec-2008, 07:13:57		
Position (km)	$R_x = -306.3$	$R_y = -42115.3$	$R_z = -2186.1$
Velocity (km/s)	$V_x = 2.7645$	$V_y = -0.3330$	$V_z = -0.1487$
Δv (km/s)	$\Delta v_r = -0.2121$	$\Delta v_t = 0.3009$	$\Delta v_n = 0.0927$

4.2.4.8 Scenario III

Scenario I is 3 impulses with $\beta = 30$, meaning the illumination angle is used. The LA determined that the first encounter time (UTC) is 18-Dec-2008 at 13:15:30. The value determined is $\Delta v = 1.7270$ km/s. Table 4.18 shows the impulses and Fig. 4.28 shows a 3D depiction of the orbits, including the transfer trajectory.

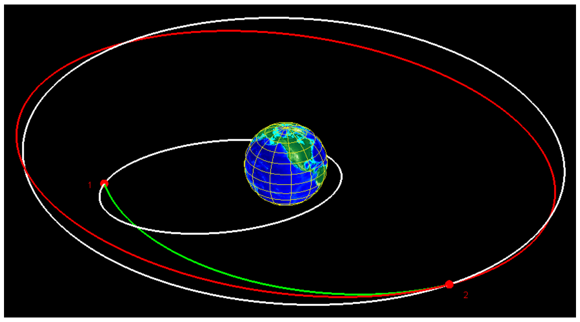

Figure 4.28: 3D view of ARIANE-44L to COSMOS-2350 PCE mission (Scenario III)

4.2.4.9 Comparison of GA, PSO, and LA

Here, the previous example, ARIANE-44L to COSMOS-2350 with 3 impulses and $\beta = 90\,\text{deg}$ (no consideration is given for the illumination constraint), is further investigated using the GA, PSO, and LA in a Monte Carlo test. For the Monte Carlo, 1000 tests are performed and the mean values are shown in Table 4.19.

Table 4.19: Monte Carlo Results of ARIANE-44L to COSMOS-2350.

Method	Mean Δv km/s	Best Δv km/s
GA	1.7729	1.6986
PSO	1.8252	1.7010
LA	1.6991	1.6955

As is seen from Table 4.19, the LA outperforms the traditional GA and PSO for this scenario.

4.2.5 Conclusions of PCE Application

Periodic Close Encounter (PCE) orbits is used for space surveillance and to better plan missions to repair damaged space assets. Previous work on PCEs have addressed the fundamental requirements and algorithms for finding optimal PCEs based on minimizing the fuel costs of the chaser satellite. The inability of those algorithms to include N-impulses, illumination, and other time constraints limited their usefulness.

In this section, it is shown how to extend the method to use N-impulses, and to include time constraints on the encounter to ensure good visibility, allowing any factor involving the target satellite's orbit that affects the observation to be accounted for. Illumination constraints are included by pre-processing the times when the illumination angle falls within a cone of size specified by the user. A minimum encounter distance is specified to optimize the observation accuracy, and to avoid direct collision. Finally, the Learning Approach to Sampling Optimization, was implemented to replace the Genetic Algorithms and shown to provide a better solution than the Genetic Algorithm and the Particle Swarm Optimization algorithm.

The next step in the development of the PCE algorithms is to include orbit perturbations, including Earth oblateness effects and atmospheric drag, as these will impact not only the encounter position and velocity, but also the compatibility of the two orbits that provides long-term coverage. Further work is also being done on implementing the ability to encounter multiple satellites. This requires the PCE orbit to be compatible with multiple other target orbits.

4.3 Spacecraft Trajectory Optimization – Cassini Mission

The Cassini-Huygens spacecraft launched on October 15, 1997 as a joint mission between NASA, ESA, and ASI to study Saturn and it's moons. With a mass of roughly 6,000 kg, Cassini entered Saturn's orbit on July 1, 2004. The design of the Cassini mission included multiple fly-bys of Venus, Earth, and Jupiter on the trajectory to Saturn in order to lower the total Δv required by the spacecraft. The fly-bys were gravity assist maneuvers, which use the gravity of a planet to alter the velocity of a spacecraft in order to save fuel and/or time. The assist is provided by the orbital angular momentum of the massive body pulling on the spacecraft as it moves near the planet. It should be noted that a Hohmann transfer from Earth to Saturn required roughly 15.7 km/s of Δ v. Both of the models shown in this section provide a much lower cost.

The dynamical models and planetary ephemerides (and their propagation) used for the study were taken from developments provided by the European Space Agency's Advanced Concepts Team[94, 95]. Two models, the MGA and the MGADSM, are here described. The objective is to launch a spacecraft from Earth and reach Saturn and be captured by its gravity in an orbit with radius of perigee, $r_p = 108,950$ km and eccentricity, $e = 0.98$. The planetary fly-by sequence is Earth-Venus-Venus-Earth-Jupiter-Saturn, which is the one used by the actual Cassini spacecraft. Constraints are given for fly-by perigees as shown in Table 4.20.

Table 4.20: Constraints of Planetary Fly-By Perigee Radius

Planet	min r_p (km)
Earth	6,778.1
Venus	6,351.8
Jupiter	600,000

4.3.1 Cassini1

The first model is an MGA, meaning *Multiple Gravity Assist* problem and is the simpler of the two models. The state is six dimensions which represent the times of launch and planetary encounter. For the Cassini1 model, the following six dimensional state vector is used to minimize the total Δv required are shown in Table 4.21.

The variables are:

- t_0, the launch date;
- T_i, $i = 1, \ldots, 6$ are the time of flight along the i^{th} leg, joining planet B_{i-1} with planet B_i

Given the values of the time variables, the positions of each planet, B_i, is computed by propagating the ephemerides. Thus, the solution of the corresponding

Table 4.21: State Variables for Cassini1

State	Variable	Lower Bound	Upper Bound	Units
x(1)	t_0	-1000	0	MJD2000
x(2)	t_1	30	400	days
x(3)	t_2	100	470	days
x(4)	t_3	30	400	days
x(5)	t_4	400	2000	days
x(6)	t_5	1000	6000	days

Lambert arcs (trajectories obtained by solving Lambert's problem) are computed along all legs. The Δv required given the solution to Lambert's problem at each leg is then computed. The spacecraft provides a single, impulsive Δv to transfer between the Lambert arcs. Therefore, the total cost function is given as

$$||\Delta v_0|| + \sum_{i=1}^{5} ||\Delta v_i|| + ||\Delta v_6|| \tag{4.53}$$

and the constraints are given in Table 4.20.

4.3.2 Cassini2

The MGADSM represents a *Multiple Gravity Assist Deep Space Maneuver* problem. This model is more flexible and realistic, but also makes the problem necessarily more complex. In addition to the time variables from Cassini1, the following variables are required:

- v_∞, u, v, the modulus and direction (two angles) of the relative velocity to the Earth at launch;
- η_i, $i = 1, \ldots, 5$, which defines when the DSM takes place on each leg;
- r_i, $i = 1, \ldots, 4$, the perigee radius at each body;

- b_i, $i = 1, \ldots, 4$, the angle of the outgoing velocity

In particular, the value of $\eta_i \in [0, 1]$ is used in the following equation

$$t_i^{DSM} = t_0 + \sum_{j=1}^{i-1} T_j + \eta_i T_i \tag{4.54}$$

This way, there are two Lambert arcs in each leg. In addition, the incoming velocity of and the orbit eccentricity at each planet (in the leg from the previous DSM) defines a cone, along whose surface the spacecraft outgoing velocity lies. The angle b_i defines where on the surface of the cone the velocity is. The addition of the DSM requires additional Δv terms in the cost function.

The problem has dimension $N = 6+4(B-2)$, where B is the number of planetary bodies. This problem is more complicated and contains 22 states. The problem is considered a rendezvous problem rather than an orbital insertion problem (as the MGA model is). Thus, the cost function in the MGADSM problem is expected to be higher than in the MGA problem. The same fly-by perigees are used (as shown in Table 4.20). Table 4.22 shows the bounds on the Cassini2 problem.

4.3.3 Results of Numerical Tests

Numerical tests are run in a Monte Carlo fashion for a series of algorithms. As with previous work[94, 95], no information on the problem (e.g., analytical derivatives) was exploited. This problem is purely a numerical solution. The algorithms tested include the following:

1. Genetic Algorithm[29]

2. Particle Swarm Optimization[31]

3. Differential Evolution[39]

Table 4.22: State Variables for Cassini2

State	Variable	Lower Bound	Upper Bound	Units
x(1)	t_0	-1000	0	MJD2000
x(2)	v_{inf}	3	5	km/s
x(3)	u	0	1	none
x(4)	v	0	1	none
x(5)	t_1	100	400	days
x(6)	t_2	100	500	days
x(7)	t_3	30	300	days
x(8)	t_4	400	1600	days
x(9)	t_5	800	2200	days
x(10)	η_1	0.01	0.9	none
x(11)	η_2	0.01	0.9	none
x(12)	η_3	0.01	0.9	none
x(13)	η_4	0.01	0.9	none
x(14)	η_5	0.01	0.9	none
x(15)	r_{p1}	1.05	6	none
x(16)	r_{p2}	1.05	6	none
x(17)	r_{p3}	1.15	6.5	none
x(18)	r_{p4}	1.7	29.1	none
x(19)	b_1	$-\pi$	π	rads
x(20)	b_2	$-\pi$	π	rads
x(21)	b_3	$-\pi$	π	rads
x(22)	b_4	$-\pi$	π	rads

4. Simulated Annealing[24]

5. Learning Approach to Sampling Optimization (as described in Chapter III)

The tests are performed 10,000 times with each algorithm allowed 100,000 cost function calls for Cassini1 and 500,000 cost function evaluations for Cassini2. The GA parameters are shown in Table 4.23, the PSO parameter values are shown in Table 4.24, and the DE parameters are shown in Table 4.25. The SA was initialized with a different random point within the search space for every Monte Carlo run, which caused the results to suffer. For the SA, the temperature schedule is started at 100 degrees and cooled to 0 degrees.

The results shown are run on an HP G71 laptop computer with 4 gigabytes of RAM and a Pentium Dual-Core CPU operating at 2.10 GHz. It should be noted that few other research groups who have investigated these problems, are forthcoming with the hardware or algorithm used. However, it is known that many of the groups have access to clusters, each with many nodes and the algorithms developed are specialized for spacecraft global trajectory problems, and are also very sensitive to initial guesses. In fact, some of the methods use a GA to produce a first guess for their optimization method. The comparisons made here are therefore in that vain, finding a suitable method as a first guess for a more complex (yet narrowly dedicated) method.

It was expected that none of the classical algorithms (GA, PSO, DE, and SA) would accurately and repeatedly find the minima Δv solution, however, these results can provide a good starting point for hybrid and/or deterministic based methods. Based on previous results, the LA was expected to outperform the more traditional methods in accuracy, repeatability, and computational speed. The statistically important data is summarized in Table 4.26 for Cassini1 and Table 4.27 for Cassini2.

Table 4.23: GA Parameters for Cassini Model Testing

Parameter	Value *Cassini1*	Value *Cassini2*
Population	500	1,000
Generations	200	500
Crossover Rate	0.6	0.6
Mutation Rate	0.2	0.2

Table 4.24: PSO Parameters for Cassini Model Testing

Parameter	Value *Cassini1*	Value *Cassini2*
Population	500	500
Generations	200	1,000
ω	0.65	0.65
η_1	2	2
η_2	2	2
$v_{\max}$	0.5	0.5

Table 4.25: DE Parameters for Cassini Model Testing

Parameter	Value *Cassini1*	Value *Cassini2*
Population	500	500
Generations	200	1,000

Table 4.26: Numerical Test Results for *Cassini1*

Algorithm	Mean J($\mathbf{x}$)	Standard Deviation on J($\mathbf{x}$)	Best J($\mathbf{x}$)	Mean CPU Time
GA	10.0853	1.0651	6.0659	336 secs
PSO	8.2454	0.9942	5.3715	382 secs
DE	12.1461	1.2923	8.1952	361 secs
SA	8.3289	0.8977	5.4935	369 secs
LA	5.0283	0.0324	4.9308	370 secs
Best Known	—	—	4.9307	—

Table 4.27: Numerical Test Results for *Cassini2*

Algorithm	Mean J($\mathbf{x}$)	Standard Deviation on J($\mathbf{x}$)	Best J($\mathbf{x}$)	Mean CPU Time
GA	29.665	0.830	25.551	568 secs
PSO	21.739	0.734	18.511	1213 secs
DE	34.284	0.650	30.863	989 secs
SA	22.239	0.413	20.298	1018 secs
LA	8.918	0.101	8.406	684 secs
Best Known	—	—	8.383	—

The mean, standard deviation, best solution, and CPU time over the series of computations are reported.

4.3.4 Conclusions of Cassini Mission Models Application

The performance of the LA as a front-end method to provide an initial guess to a more complex method is shown using two models representative of the Cassini spacecraft trajectory to Saturn. The LA is shown to perform best when compared with the other optimization algorithms in terms of accuracy (i.e., lowest cost function) and is comparable in terms of evaluation speed. This application additionally extended the LA to a 22-dimensional problem and shows promise that the method can be applied in even higher dimensions and on more complex cost functions.

5. CONCLUSIONS

In this dissertation, a new numerical optimization technique was developed and tested. The purpose of the Learning Approach to Sampling Optimization algorithm was to be a general numerical optimization algorithm similar to the popular Genetic Algorithm. However, the LA has fewer parameters to tune and was designed for simple implementation (e.g., on a laptop instead of a cluster). The general theory and a mathematical proof for a special case were presented. A study of performance against common benchmark functions and popular numerical optimization algorithms was presented. Then, the LA was applied to problems in astrodynamics. A set of analytical bounds used in orbit transfer and rendezvous Δv selection was derived. Finally, the LA was applied to higher-dimensional problems of orbit transfer. It was shown through the examples that the LA algorithm is capable of generating results comparable to popular numerical optimization methods.

Overall, the work presented in this dissertation accomplished the goals set out in the Introduction section:

1. To develop a numerical optimization method that is based on a sound mathematical foundation;
2. To compare the performance of the algorithm with currently used algorithms;
3. To pose new problems in the field of astrodynamics;
4. To solve the newly posed problems and compare the solutions with other algorithms where possible.

The basis of the LA algorithm lies in rejection sampling whereby points are accepted into the data set based on the current estimate of the probability density function. The theory was developed in a general way, not restricted by dimension. By comparison to the random search for a simple function, a mathematical proof was presented showing that, statistically, the LA converged faster for an arbitrary accuracy. A numerical test backed up the proof. Finally a set of benchmark functions from the computer science literature were tested. While most of these were two-dimensional functions, they have value in showing the LA algorithm and give positive preliminary results. The LA was shown to give excellent results for multi-minimum functions (similar to those seen in astrodynamics and many other fields) as well as discontinuous functions. Modifications for quicker convergence were then presented, including allowing the LA to be adaptive (time-varying) as well as parallelized. The log-normal distribution showed great promise, but had the drawback of introducing a new parameter to tune.

While examining orbit transfer problems in astrodynamics, it was realized that many optimization problems fail because of a lack of proper bounds. As such, a set of analytical bounds was developed for the selection of Δv values such that the new orbit neither intersected the Earth (plus a given height of atmosphere) nor became a parabolic or hyperbolic trajectory away from Earth. The N-impulse orbit transfer and rendezvous problems were posed in a way conducive to numerical optimization. Periodic Close Encounters were shown to reduce Δv requirements while maintaining user-defined observation metrics. Constraints were introduced to maximize observation clarity.

Finally, three major applications from astrodynamics were solved with positive results. The N-impulse orbit transfer and rendezvous problem was solved numerically. The results matched well with published and analytical solutions and new solutions were found. Periodic Close Encounters were introduced, which is a particular case of orbit transfers. Constraints were imposed in the optimization problem and multiple examples were solved. Two test cases from the European Space Agency, representing different fidelity models of the *Cassini* trajectory, were also tested. While the results of the LA did not produce a globally optimal result, the cost function was relatively small compared to the currently best known cost function value. The *Cassini* results showed the LA performed well in a high-dimensional problem and the results were more repeatable than other numerical optimization algorithms.

This leads the author to believe that the LA algorithm would serve well as the front-end optimization algorithm for other local search algorithms. For example, the LA could be used to determine regions of interest and the PDF within that region. Then, other methods could be used to perform the local search within the regions of interest. Of concern with the LA is finding the nearest neighbors in higher dimensions and fitting the hyper-plane between the neighbors. Database methods were out of the scope of the current work, but should be investigated as the LA is used in more applications of high dimension to speed up the computations.

REFERENCES

[1] J. Gomulka, *Towards Global Optimisation 2.* New York, NY: North-Holland Publishing Company, 1978, ch. Deterministic vs Probabilistic Approaches to Global Optimisation, pp. 19–29.

[2] S. S. Rao, *Engineering Optimization: Theory and Practice*, 3rd ed. New Delhi: New Age International (P) Limited, 1996.

[3] J. A. Lozano, *Estimation of Distribution Algorithms*, ser. Genetic Algorithms and Evolutionary Computation. Norwell, MA: Kluwer Academic Publishers, 2002, ch. An Introduction to Evolutionary Algorithms, pp. 3–25.

[4] T. Bäck, U. Hammel, and H.-P. Schwefel, *Evolutionary Computation: The Fossil Record.* New York, NY: IEEE Press, 1998, ch. Evolutionary Computation: Comments on the History and Current State.

[5] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical Optimization: Theoretical and Practical Aspects.* New York: Springer-Verlag, 2003.

[6] T. Bäck, *Evolutionary Algorithms in Theory and Practice.* Oxford, UK: Oxford University Press, 1996.

[7] L. C. W. Dixon and G. P. Szegö, *Towards Global Optimisation 2.* New York, NY: North-Holland Publishing Company, 1978, ch. The Global Optimisation Problem: An Introduction, pp. 1–15.

[8] C. G. E. Boender, A. H. G. Rinnooy Kan, and G. T. Timmer, “A Stochastic Method for Global Optimization,” *Mathematical Programming*, vol. 22, pp. 125–140, 1982.

[9] A. Törn and A. Žilinskas, *Global Optimization*, ser. Lecture Notes in Computer Science, G. Goos and J. Hartmanis, Eds. New York: Springer-Verlag, 1989, vol. 350.

[10] R. E. Bellman, *Adaptive Control Processes: A Guided Tour.* Princeton, NJ: Princeton University Press, 1961.

[11] R. Salomon, “Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions - A survey of some theoretical and practical aspects of genetic algorithms,” *BioSystems*, vol. 39, pp. 263–278, 1995.

[12] T. Vinkó and D. Izzo, “Global Optimisation Heuristics and Test Problems for Preliminary Spacecraft Trajectory Design,” European Space Agency, Advanced Concepts Team, Noordwijk, The Netherlands, Tech. Rep. ACT-TNT-MAD-GOHTPPSTD, September 2008.

[13] A. J. Keane, *Theoretical Aspects of Evolutionary Computing*, ser. Theoretical Aspects of Evolutionary Computation. New York: Springer-Verlag, 2001, ch. An Introduction to Evolutionary Computing in Design Search and Otimisation, pp. 1–11.

[14] J. J. Grefenstette, “Optimization of Control Parameters for Genetic Algorithms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.

[15] P. Larrañaga, *Estimation of Distribution Algorithms*, ser. Genetic Algorithms and Evolutionary Computation. Norwell, MA: Kluwer Academic Publishers, 2002, ch. A Review on Estimation of Distribution Algorithms, pp. 57–100.

[16] A. E. Eiben, *Theoretical Aspects of Evolutionary Computing*, ser. Theoretical Aspects of Evolutionary Computation. New York: Springer-Verlag, 2001, ch. Evolutionary Algorithms and Constraint Satisfaction: Definitions, Survey, Methodology, and Research Directions, pp. 13–30.

[17] D. B. Fogel, *Evolutionary Algorithms in Engineering and Computer Science.* New York, NY: John Wiley & Sons, LTD, 1999, ch. An Introduction to Evolutionary Computation and Some Applications.

[18] ——, *Evolutionary Algorithms in Engineering and Computer Science.* New York, NY: John Wiley & Sons, LTD, 1999, ch. Some Recent Important Foundational Results in Evolutionary Computation.

[19] H. Robbins and S. Monro, "A Stochastic Approximation Method," *Annals of Mathematical Stastics*, vol. 22, pp. 400–407, 1951.

[20] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution.* New York: Wiley, 1966.

[21] J. Haataja, *Evolutionary Algorithms in Engineering and Computer Science.* New York, NY: John Wiley & Sons, LTD, 1999, ch. Using Genetic Algorithms for Optimization: Technology Transfer in Action.

[22] W. K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[23] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The University of Michigan Press, 1975.

[24] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.

[25] F. Schoen, "Stochastic Techniques for Global Optimization: A Survey of Recent Advanaces," *Journal of Global Optimization*, vol. 1, pp. 207–228, 1991.

[26] A. M. Browder, "A New Framework for Global Optimization: Applications to Control of Flexible Structures," PhD Department of Aerospace Engineering, Texas A&M University, College Station, TX, December 1993.

[27] F. J. Solis and R. J.-B. Wets, "Minimization by Random Search Techniques," *Mathematics of Operations Research*, vol. 6, pp. 19–30, 1981.

[28] S. H. Brookes, "A Discussion of Random Methods for Seeking Maxima," *Operations Research*, vol. 6, pp. 244–251, March–April 1958.

[29] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[30] D. B. Fogel, *Evolutionary Computation: The Fossil Record.* New York, NY: IEEE Press, 1998, ch. An Introduction to Simulated Evolutionary Optimization.

[31] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, Perth, Australia, 1995, pp. 1942–1948.

[32] G. Venter and J. Sobieski, "Particle Swarm Optimization," in *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, CO, April 2002.

[33] S. M. Mikki and A. A. Kishk, *Particle Swarm Optimization: a physics-based approach*, ser. Synthesis Lectures on Computational Electromagnetics. San Rafael, CA: Morgan and Claypool, 2008, vol. 20.

[34] G. Venter and J. Sobieszczanski-Sobieski, "Particle Swarm Optimization," *AIAA Journal*, vol. 41, no. 8, pp. 1583–1589, 2003.

[35] M. Clerc and J. Kennedy, "The Particle Swarm - Explosion, Stability and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, February 2002.

[36] J. Kennedy, "The Particle Swarm: Social Adaptation of Knowledge," in *International Conference on Evolutionary Computation*, Indianapolis, IN, April 1997, pp. 303–308.

[37] J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*, ser. The Morgan Kaufmann Series in Evolutionary Computation. San Francisco: Morgan Kaufmann, 2001.

[38] R. Storn and K. Price, "Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[39] A. Olds, C. A. Kluever, and M. Cupples, "Interplanetary Mission Design Using Differential Evolution," *Journal of Spacecraft and Rockets*, vol. 44, no. 5, pp. 1060–1070, 2007.

[40] R. Storn, "On the Usage of Differential Evolution for Function Optimization," in *1996 Biennial Conference of the North American Fuzzy Information Processing Society*, Berkeley, CA, 1996, pp. 519–523.

[41] R. Biesbroek, “A Comparison of Differential Evolution Method with Genetic Algorithms for Orbit Optimisation,” in *International Astronautical Federation*, Valencia, Spain, October 2–6 2006.

[42] P. Lu and M. Kahn, “Nonsmooth Trajectory Optimization - An Approach Using Continuous Simulated Annealing,” *Journal of Guidance, Control, and Dynamics*, vol. 17, no. 4, pp. 685–691, 1994.

[43] O. Tekinalp and M. Bingol, “Simulated Annealing for Missile Optimization: Developing Method and Formulation Techniques,” *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 4, pp. 616–626, July–August 2004.

[44] N. E. Collins, R. W. Eglese, and B. L. Golden, “Simulated Annealing–An Annotated Bibliography,” *American Journal of Mathematical and Management Sciences*, vol. 8, pp. 209–307, January 1988.

[45] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equations of State Calculations by Fast Computing Machines,” *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[46] H. Mühlenbein and G. Paaß, “From recombination of genes to the estimation of distributions i. binary parameters,” in *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature*. Springer-Verlag, 1996, pp. 178–187.

[47] C. González, J. A. Lozano, and P. L. naga, *Estimation of Distribution Algorithms*, ser. Genetic Algorithms and Evolutionary Computation. Norwell, MA: Kluwer Academic Publishers, 2002, ch. Mathematical Modeling of Discrete Estimation of Distribution Algorithms, pp. 147–163.

[48] R. H. Battin, *An Introduction to the Mathematics and Methods of Astrodynamics*, revised ed. New York: AIAA Education Series, 1987.

[49] D. A. Vallado, *Fundamentals of Astrodynamics and Applications.* McGraw–Hill, New York, 2001, vol. 2.

[50] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems.* Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2003.

[51] R. H. Gooding, "A Procedure for the Solution of Lambert's Orbital Boundary-Value Problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 48, no. 2, 1990.

[52] J. von Neumann, "Various Techniques used in Connection with Random Digits. Monte Carlo Methods," *Nat. Bureau Standards, Appl. Math. Ser.*, vol. 12, pp. 36–38, 1951.

[53] D. B. Fogel, "An Introduction to Evolutionary Computation," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3–14, 1994.

[54] T. A. Henderson, D. Mortari, M. E. Avendaño, and J. L. Junkins, "An Adaptive and Learning Approach to Sampling Optimization," in *19th AAS/AIAA Spaceflight Mechanics Meeting*, Savannah, Georgia, February 2009.

[55] T. A. Henderson and D. Mortari, "A Learning Approach to Sampling Optimization Applied to a Global Trajectory Optimization Problem," in *2009 AAS/AIAA Astrodynamics Specialist Conference*, Pittsburgh, PA, August 2009.

[56] T. A. Henderson, D. Mortari, and M. E. Avendaño, "A Learning Approach to Sampling Optimization," 2010.

[57] MathWorks, "Documentation of Genetic Algorithm and Direct Search Toolbox," Online, http://www.mathworks.com/access/helpdesk/help/toolbox/gads/gads.shtml.

[58] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, "A Comparison of Particle Swarm Optimization and the Genetic Algorithm," in *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Austin, TX, April 2005.

[59] R. C. Eberhart and Y. Shi, "Comparison Between Genetic Algorithms and Particle Swarm Optimization," in *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, San Diego, CA, 1998.

[60] J. Olvander and P. Krus, "Optimizing the Optimization. A Method for Comparison of Optimization Algorithms," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Newport Rhode Island, May 2006.

[61] H. H. Rosenbrock, "An Automatic Method for Finding the Greatest or Least Value of a Function," *Computer Journal*, vol. 3, pp. 175–184, 1960.

[62] D. H. Ackley, *A Connectionist Machine for Genteic Hillclimbing.* Norwell, MA: Kluwer Academic Press, 1987.

[63] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gümüs, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger, *Handbook of Test Problems in Local and Global Optimization.* Boston: Kluwer Academic Publishers, 1999.

[64] L. Devroye, *Non-Uniform Random Variate Generation.* New York: Springer-Verlag, 1986.

[65] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," in *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases.* San Francisco: Morgan Kaufmann Publishers Inc., 1998, pp. 194–205.

[66] B. B. Spratling IV, "Recursive Star Identification with the K-Vector ND," in *2010 AAS/AIAA Space Flight Mechanics Meeting*, San Diego, CA, February 2010.

[67] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin: Springer, 2008.

[68] J. J. Davis, "Constellation Reconfiguration: Tools & Analysis," PhD Department of Aerospace Engineering, Texas A&M University, College Station, TX, May 2010.

[69] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[70] D. T. Lee and C. K. Wong, "Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees," *Acta Informatica*, vol. 9, no. 1, pp. 23–29, March 1977.

[71] L. K. Grover, "From Schrödinger's Equation to Quantum Search Algorithm," *American Journal of Physics*, vol. 69, no. 7, pp. 769–777, July 2001.

[72] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[73] F. Cacciatore and C. Toglia, "Optimization of Orbital Trajectories Using Genetic Algorithms," *Journal of Aerospace Engineering, Sciences and Applications*, vol. 1, no. 1, pp. 58–69, January–April 2008.

[74] W. Hohmann, *Die Erreichbarkeit der Himmelskörper.* München, Germany: Verlag Oldenbourg, 1925.

[75] C. Marchal, "Transferts Optimaux Entre Orbites Elliptiques Coplanaires (Durée Indifférente)," *Astronautica Acta*, vol. 11, no. 6, pp. 432–445, November–December 1965.

[76] R. F. Hoelker and R. Silber, "The Bi-Elliptic Transfer Between Circular Co-Planar Orbits," Army Ballistic Missile Agency, Redstone Arsenal, Tech. Rep. 2-59, January 1959.

[77] D. F. Lawden, *Optimal Trajectories for Space Navigation.* London: Butterworths, 1963.

[78] J. E. Prussing, "Optimal Two- and Three-Impulse Fixed-Time Rendezvous in the Vicinity of a Circular Orbit," *Journal of Spacecraft and Rockets*, vol. 40, no. 6, pp. 952–959, November–December 2003.

[79] D. J. Jezewski and H. L. Rozendaal, "An Efficient Method for Calculating Optimal Free-Space N-Impulse Trajectories," *AIAA Journal*, vol. 6, no. 11, pp. 2160–2165, November 1968.

[80] G. Colasurdo and D. Pastrone, "Indirect Optimization Method for Impulsive Transfers," in *AIAA/AAS Astrodynamics Conference*, Scottsdale, AZ, August 1994, pp. 441–448.

[81] J. E. Prussing, "Equation for Optimal Power-Limited Spacecraft Trajectories," *Journal of Guidance, Control, and Dynamics*, vol. 16, no. 2, pp. 391–393, 1993.

[82] P. J. Gage, R. D. Braun, and I. M. Kroo, "Interplanetary Trajectory Optimization Using a Genetic Algorithm," *Journal of the Astronautical Sciences*, vol. 43, no. 1, pp. 59–75, 1995.

[83] G. A. Rauwolf and V. L. Coverstone-Carroll, "Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm," *Journal of Spacecraft and Rockets*, vol. 33, no. 6, pp. 859–862, November–December 1996.

[84] F. Gobetz and J. Doll, "A Survey of Impulsive Trajectories," *AIAA Journal*, vol. 7, no. 5, pp. 801–834, 1969.

[85] Y. H. Kim and D. B. Spencer, "Optimal Spacecraft Rendezvous Using Genetic Algorithms," *Journal of Spacecraft and Rockets*, vol. 39, no. 6, pp. 859–865, November–December 2002.

[86] O. O. Abdelkhalik and D. Mortari, "n-Impulse Orbit Transfer Using Genetic Algorithms," *Journal of Spacecraft and Rockets*, vol. 44, no. 2, pp. 456–459, March–April 2007.

[87] D. Mortari, T. Henderson, J. Davis, and M. Avendaño, "Satellite Constellation Design Optimization Methods Study," Texas A&M University, College Station, Tech. Rep., September 2009, aFRL Contract FA9453-06-C-0342.

[88] T. A. Henderson, D. Mortari, and M. E. Avendaño, "Admissible n-impulse Orbit Transfer and Rendezvous Solved Using a Learning Optimization Algorithm," in *AAS 10–252 of the 20th AAS/AIAA Space Flight Mechanics Meeting Conference*, San Diego, CA, February 8–12 2010.

[89] K. A. De Jong, "Analysis of the Behavior of a Class of Genetic Adaptive Systems," PhD Department of Computer Science, University of Michigan, Ann Arbor, MI, May 1975.

[90] A. Clocchiatti and D. Mortari, "Responsive Space Surveillance using Periodic Close Encounters," in *7th Dynamics and Control of Systems and Structures in Space Conference*, July 2006.

[91] A. Clocchiatti, "Responsive Space Surveillance using Periodic Close Encounters," MS Thesis, Ingegneria Aerospaziale, Politecnico di Milano, July 2006.

[92] O. O. Abdelkhalik and D. Mortari, "Orbit Design for Ground Surveillance Using Genetic Algorithms," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 5, pp. 1231–1235, September–October 2006.

[93] ——, "Reconnaissance Problem Using Genetic Algorithms," in *Paper AAS 05–184 of the 2005 Space Flight Mechanics Meeting Conference*, Copper Mountain, Colorado, January 23–27 2005.

[94] T. Vinkó, D. Izzo, and C. Bombardelli, "Benchmarking different global optimisation techniques for preliminary space trajectory design," in *58th International Astronautical Congress*, Hyderabad, India, September 24–26 2007.

[95] D. Izzo and T. Vinkó, "Act-informatics," May 2010, `http://www.esa.int/gsp/ACT/inf/op/globopt.htm`.