The Pennsylvania State University

The Graduate School

College of Engineering

EVOLUTIONARY COMPUTATION FOR SPACECRAFT TRAJECTORY OPTIMIZATION

A Thesis in

Aerospace Engineering

by

Bradley Joseph Sottile

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2013

The thesis of Bradley Joseph Sottile was reviewed and approved* by the following:

Robert G. Melton
Professor of Aerospace Engineering
Thesis Advisor

David B. Spencer
Professor of Aerospace Engineering

George A. Lesieutre
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Evolutionary Computation has exploded in use in engineering and the applied sciences. For this thesis, three algorithms – Particle Swarm Optimization (PSO), Bacteria Foraging Optimization (BFO) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) – are compared against each other to solve a classic problem in astrodynamics, the Hohmann transfer. The role of fixed and varying penalties is explored for each algorithm and compared. Each algorithm was run 1000 times and the performance metrics were compared. PSO using fixed penalties ran with an average central processing unit (CPU) time of 0.138 seconds and yielded a mean error of 1.30% and a median error of 0.48%. Using varying penalties, the algorithm ran with an average CPU time of 0.107 seconds and yielded a mean error of 1.78% and a median error of 0.43%. BFO with fixed penalties had a mean CPU time of 0.655 seconds and yielded a 2.19% mean percent error and 1.91% median percent error. For the varying penalty case, BFO averaged a CPU time of 0.727 seconds, a mean percent error of 0.27% and a median 0.36%. CMA-ES with fixed penalties yielded a mean CPU time of 0.572 seconds, a mean percent error of 0.26% and a median percent error of 0.00%. The varying penalty case for CMA-ES yielded a mean CPU time of 0.582 seconds, a mean percent error of 0.43% and a median percent error of 0.43%. The algorithms all excelled in some areas and had poor performance in others, especially as the penalty case varied. A clear result is that algorithm selection is problem-dependent. Suggestions for future work and applications to other problems are provided.

TABLE OF CONTENTS

List of Figures v

List of Tables vii

Acknowledgements viii

Chapter 1 Introduction 1

1.1 Thesis Motivation 1
1.2 A Brief Overview of Evolutionary Computation 2
1.3 Thesis Overview and Organization 3

Chapter 2 Evolutionary Computation and Algorithm Descriptions 4

2.1 Particle Swarm Optimization (PSO) 4
2.2 Bacterial Foraging Optimization (BFO) 6
2.3 Covariance Matrix Adaptation Evolution Strategy (CMA-ES) 8

Chapter 3 Optimal Two-Impulse Transfers Between Circular Orbits 12

3.1 Hohmann Transfer background 12
3.2 Problem Statement 18
3.3 Numerical Implementation 21

Chapter 4 Results and Discussion 23

4.1 Numerical Results 23
4.2 Discussion 47

Chapter 5 Conclusions and Future Work 49

5.1 Lessons Learned 49
5.2 Future Work 50

References 51

LIST OF FIGURES

Figure 4.1. Fitness vs. Iterations of β = 2 for a Single Run of PSO using Fixed Penalties 23

Figure 4.2. Fitness vs. Iterations of Various β for Single Runs of PSO using Fixed Penalties 24

Figure 4.3. Detailed View of Fitness vs. Iterations of Various β for Single Runs of PSO using Fixed Penalties.....25

Figure 4.4. Box Plot of β = 2 for 1000 Runs of PSO with Fixed Penalties for Percent Error 26

Figure 4.5. Box Plot of β = 2 for 1000 Runs of PSO with Fixed Penalties for Time Elapsed.....26

Figure 4.6. Fitness vs. Iterations of β = 2 for a Single Run of PSO using Varying Penalties 28

Figure 4.7. Detailed View of Fitness vs. Iterations of β = 2 for a Single Run of PSO using Varying Penalties.....28

Figure 4.8. Fitness vs. Iterations of Various β for Single Runs of PSO using Varying Penalties 29

Figure 4.9. Detailed View of Fitness vs. Iterations of Various β for Single Runs of PSO using Varying Penalties.....30

Figure 4.10. Box Plot of β = 2 for 1000 Runs of PSO with Varying Penalties for Percent Error 31

Figure 4.11. Box Plot of β = 2 for 1000 Runs of PSO with Varying Penalties for Time Elapsed.....31

Figure 4.12. Fitness vs. Iterations of β = 2 for a Single Run of BFO using Fixed Penalties 32

Figure 4.13. Fitness vs. Iterations of Various β for Single Runs of BFO using Fixed Penalties.....33

Figure 4.14. Detailed View of Fitness vs. Iterations of Various β for Single Runs of BFO using Fixed Penalties.....33

Figure 4.15. Box Plot of β = 2 for 1000 Runs of BFO with Fixed Penalties for Percent Error.....34

Figure 4.16. Box Plot of β = 2 for 1000 Runs of BFO with Fixed Penalties for Time Elapsed 35

Figure 4.17. Fitness vs. Iterations of β = 2 for a Single Run of BFO using Varying Penalties 36

Figure 4.18. Fitness vs. Iterations of Various β for Single Runs of BFO using Varying Penalties.....37

Figure 4.19. Box Plot of β = 2 for 1000 Runs of BFO with Varying Penalties for Percent Error.....38

Figure 4.20. Box Plot of β = 2 for 1000 Runs of BFO with Varying Penalties for Time Elapsed 38

Figure 4.21. Fitness vs. Iterations of β = 2 for a Single Run of CMA-ES using Fixed Penalties.....39

Figure 4.22. Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Fixed Penalties.....40

Figure 4.23. Detailed View of Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Fixed Penalties....40
Figure 4.24. Box Plot of β = 2 for 1000 Runs of CMA-ES with Fixed Penalties for Percent Error....41
Figure 4.25. Box Plot of β = 2 for 1000 Runs of CMA-ES with Fixed Penalties for Time Elapsed....42
Figure 4.26. Fitness vs. Iterations of β = 2 for a Single Run of CMA-ES using Varying Penalties....43
Figure 27. Detailed View of Fitness vs. Iterations of β = 2 for a Single Run of CMA-ES using Varying Penalties....43
Figure 4.28. Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Varying Penalties....44
Figure 4.29. Detailed View of Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Varying Penalties....44
Figure 4.30. Box Plot of β = 2 for 1000 Runs of CMA-ES with Varying Penalties for Percent Error....45
Figure 4.31. Box Plot of β = 2 for 1000 Runs of CMA-ES with Varying Penalties for Time Elapsed....46

LIST OF TABLES

Table 3.1. Range of β Values and Effect on Optimality....13

Table 4.1. Performance Metrics of β = 2 for a Single Run of PSO using Fixed Penalties....24

Table 4.2. Performance Metrics of Various β for a Single Run of PSO with Fixed Penalties....25

Table 4.3. Performance Metrics of β = 2 for 1000 Runs of PSO using Fixed Penalties....27

Table 4.4. Performance Metrics of β = 2 for a Single Run of PSO using Varying Penalties....29

Table 4.5. Performance Metrics of Various β for a Single Run of PSO with Varying Penalties....30

Table 4.6. Performance Metrics of β = 2 for 1000 Runs of PSO using Varying Penalties....32

Table 4.7. Performance Metrics of β = 2 for a Single Run of BFO using Fixed Penalties....33

Table 4.8. Performance Metrics of Various β for a Single Run of BFO with Fixed Penalties....34

Table 4.9. Performance Metrics of β = 2 for 1000 Runs of BFO using Fixed Penalties....35

Table 4.10. Performance Metrics of β = 2 for a Single Run of BFO using Varying Penalties....36

Table 4.11. Performance Metrics of Various β for a Single Run of BFO with Varying Penalties....37

Table 4.12. Performance Metrics of β = 2 for 1000 Runs of BFO using Varying Penalties....39

Table 4.13. Performance Metrics of β = 2 for a Single Run of CMA-ES using Fixed Penalties....40

Table 4.14. Performance Metrics of Various β for a Single Run of CMA-ES with Fixed Penalties....41

Table 4.15. Performance Metrics of β = 2 for 1000 Runs of CMA-ES using Fixed Penalties....42

Table 4.16. Performance Metrics of β = 2 for a Single Run of CMA-ES using Varying Penalties....44

Table 4.17. Performance Metrics of Various β for a Single Run of CMA-ES with Varying Penalties....45

Table 4.18. Performance Metrics of β = 2 for 1000 Runs of CMA-ES using Varying Penalties....46

ACKNOWLEDGEMENTS

Several people made contributions to this work that I would be remiss to neglect mentioning. I would like to thank Dr. Robert G. Melton for his mentorship over the last couple of years. Without his patience and gentle advice, this project would have never come to completion. In addition, Dr. David B. Spencer has also been a great mentor as I have walked the sometimes winding road of graduate education. I would also like to thank my collaborator, Daniel Rueda, for the stimulating conversations about Evolutionary Computation and his help in the code debugging process. Many others, including Philip Myers and Michael Policelli, made contributions small and large and have been great friends as I have undertaken this work; I have spent more late nights with them than I care to remember. Finally, I would like to thank all of the friends and family who have patiently suffered through the development of this thesis; your support means more than you will ever know.

Chapter 1

Introduction

The field of astrodynamics is several hundred years old. Over that period of time, many significant advances have been made that have advanced general knowledge in the field. Giants such as Johannes Kepler (Kepler's laws), Isaac Newton (*Philosophiæ Naturalis Principia Mathematica*), Leonard Euler (numerous contributions in the mathematics and the methods of astrodynamics) and Joseph-Louis Lagrange (Lagrange points), among others, have paved the way for more recent advances by those such as Albert Einstein (relativity) and Walter Hohmann (the Hohmann Transfer). This thesis seeks to make a contribution to this large body of work as well, particularly in the area of optimization.

1.1 Thesis Motivation

Optimization is one of the oldest problems to plague mankind. A primitive example of an optimization problem (though one many people still tackle on a daily basis) is to find the shortest distance or path between two points. This problem may be constrained in some way – for example, one may only be able to travel on public roads in order to reach a given destination. Another problem might be to find the largest volume possible that could be folded from a finite area of paper. Constraints, of course, greatly affect the optimization process and outcome. In engineering, optimization is an overarching goal. For aerospace engineers, an optimization problem may take the form of minimizing the weight of an aircraft's frame without losing strength, or designing the optimal shape of an airfoil to produce the best lift to drag ratio.

Aerospace vehicle trajectory optimization is an important problem that has been studied in both aircraft and spacecraft. This thesis restricts its attention to spacecraft trajectory optimization. An important question to resolve is what specifically is to be optimized. One may desire in practice to minimize both time of flight and propellant use. In the problem presented here, the single objective goal is to minimize the change in spacecraft velocity (Δv) needed for a particular trajectory. This corresponds to a minimization of propellant use regardless of what propellant formulation is used. By minimizing the amount of propellant needed for a given mission, one is directly able to reduce cost.

1.2 A Brief Overview of Evolutionary Computation

Recently, the use of Evolutionary Computation has exploded in use in the applied sciences and engineering. In this thesis, two branches of Evolutionary Computation – Evolutionary Algorithms and Evolutionary Strategies – are explored. In their most basic formulations, Evolutionary Algorithms and Evolutionary Strategies are population-based heuristic search techniques that often model physical or biological processes such as mating, mutation or swarm behavior. Evolutionary Algorithms and Evolutionary Strategies are useful tools when one is unwilling or unable to employ the traditional gradient-based optimization techniques. Like many other disciplines in engineering, the use of Evolutionary Computation has found a home in astrodynamics – particularly in areas such as the optimization of spacecraft trajectories or in spacecraft attitude dynamics.

1.3 Thesis Overview and Organization

Chapter 2 provides an overview of evolutionary computation, with particular emphasis on evolutionary algorithms and evolutionary strategies. Detailed descriptions of the algorithms use in this work are presented. Chapter 3 describes the Hohmann transfer, proves a proof of the Hohmann transfer and presents the classic problem of the optimal two-impulsive transfer between two circular, co-planar orbits. Chapter 4 presents the numeric results from the problem developed in Chapter 3 and discusses the implications of those results. Finally, Chapter 5 discusses the conclusions of this work based on the results from the algorithms and presents suggestions for future work.

Chapter 2

Evolutionary Computation and Algorithm Descriptions

Evolutionary Computation is a relatively recent phenomenon in the literature. Aided by [1], it is possible to present a brief overview of the early history of evolutionary computation. The origins of Evolutionary Computation trace back to the late 1950's; examples from this era include [2], [3], [4] and [5]. Mostly due to the lack of processing power, this area was largely neglected early on in the literature. Eventually, during the 1970's, this situation began to change, and the foundational works by Holland [6], Rechenberg [7], Schwefel [8], and Fogel [9] lead to a shift and a renewal of interest in Evolutionary Computation. Assuming Moore's Law [10] continues to hold true, one could reasonably expect interest in Evolutionary Computation to continue unabated into the near future. This chapter introduces three specific algorithms and describes them at length. This development lays the foundation for the practical implementation of these algorithms seen later in Chapter 4. It is generally accepted that the principal differences between Evolutionary Strategies (ESs) and Evolutionary Algorithms (EAs) are that ESs: 1.) employ search steps that are deterministic and 2.) almost always work with vectors of real numbers that are representations of the solution; EAs use stochastic processes coupled with selection to produce ever-improving potential solutions. A further difference is in the notation used by the two classes of techniques.

2.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) was first introduced by James Kennedy and Russell Eberhart in 1995 [11] and is classified as a swarm intelligence method. PSO is an easy to use algorithm, in part, because there are very few parameters that need to be adjusted. A very simple

description of PSO is given by Poli [12]. Inspired by flocking behavior, a number of particles (each particle is an array of parameter values that constitutes a possible solution) placed in a parameter space of some problem. The fitness function (also known as the objective or cost function) then evaluates the fitness at that particle's location in the solution space. Based on its history, and the fitness of other particles, the particle then moves through the parameter space with a velocity determined from the fitness of other particles, along with a perturbing effect. In this sense, the particles interact with each other much like a flock of birds would interact with each other while searching for food.

For an *N* dimensional problem, the position and velocity of each particle are represented as a vector with *N* elements. The velocity vector is then given by

$$v_i(t+1) = \omega v_i(t) + \psi_1\phi_1\left(x_{s_i} - x_i(t)\right) + \psi_2\phi_2\left(x_{p_i} - x_i(t)\right) \tag{2.1}$$

where x_{s_i} is the *i*th component of the best point visited by the particles neighbors, $x_i(t)$ is the *i*th component of the particle's location, x_{p_i} is the *i*th component of the particle's best value, ϕ_1 and ϕ_2 are two independent random variables uniformly distributed ϵ $[0, 1]$, ω is a constant known as the inertia weight [13], and ψ_1 and ψ_2 are the acceleration coefficients. The acceleration coefficients control the relative proportion of social interaction in the swarm. This formula is applied to all particles. The position of the particle is then updated every time step via

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.2}$$

After all of the particles have moved, the next iteration occurs. For a more extensive discussion of the PSO algorithm, see [14].

An extensive survey was done reviewing publications on the applications of PSO [12]. 5.8% of papers in the IEEE Xplore database at that time using PSO were in the area of antenna design. Likewise, 4.8% had to do with biomedical applications, 4.4% had to do with communication networks and 7.0% had to do with the area of control applications. Design was

not left out either – 4.4% percent of papers were design-oriented. Engines and motors constituted 1.4%, entertainment (including music generation and games) consisted of another 1.4% and scheduling problems (5.6%) were also examined. Clearly, PSO has been applied in many areas of interest across various engineering disciplines.

2.2 Bacterial Foraging Optimization (BFO)

Passino first proposed Bacterial Foraging Optimization (BFO) in [15]. In his original paper, he described the biology and physics underlying the foraging (chemotactic) behavior of the *Escherichia coli* (*E. coli*) bacteria. *E. Coli* occur naturally as part of the normal flora of the gastrointestinal system and produce vitamin K_2 [16]. While they have a reputation for being harmful, *E. coli* is a large and diverse strain of bacteria and the vast majority of strains are harmless. They are readily found in the environment, in food products and in the intestines of humans and animals [17].

The following description is from [18], while the original formulation of BFO can be found in [19]. Define $\boldsymbol{u}_i$ as a random vector with uniformly distributed elements $\in [-1, 1]$ with $|\boldsymbol{u}_i| = 1$. Further define $\boldsymbol{B}$ to be a vector of parameters known as a bacterium. One can initialize a population of N bacteria set between some upper and lower limit bounds. Set the number of swim steps N_S during chemotaxis. Chemotaxis is the process by which bacteria direct their movements in order to find nutrients. Now, set the swim step size to be $C_0 = \frac{K}{N_S}$. The value of K is chosen to be the maximum allowable change in any element of $\boldsymbol{B}$ during one iteration. The condition for elimination and dispersal is $P_{ed} \in [0, 1]$; this condition is important because it reduces the probability that stagnation will occur. Finally, define r to be a random number uniformally distributed $\in [0, 1]$. Noting that J represents the result of the fitness function evaluation, it is then possible to write the following pseudocode:

for $k = 1: N_{iter}$ **do**

Tumble and Swim (Chemotaxis)
for $i = 1: N$ **do**
Generate $\boldsymbol{u}_i$
$\boldsymbol{B}_i' = \boldsymbol{B}_i + C(k)\,\boldsymbol{u}_i$, a trial bacterium
$C(k) = C_0 - \frac{0.75C_0(k-1)}{N_{iter}}$
Set the step counter $m = 0$
While $J(\boldsymbol{B}_i') < J(\boldsymbol{B}_i)$ and $m \leq N_S$ **do**
$\boldsymbol{B}_i = \boldsymbol{B}_i'$
$\boldsymbol{B}_i' = \boldsymbol{B}_i + C(k)\,\boldsymbol{u}_i$
$m = m + 1$
end while
end for

Reproduction
Sort the bacteria in ascending order of J. Replace the worst $\frac{N}{2}$ bacteria (*i.e.* those with the highest J values) with copies of the best $\frac{N}{2}$ bacteria

Elimination and Dispersal
for $i = 1: N$ **do**
if $r < P_{ed}$ **then**
replace B_i with a randomly generated bacterium within a specified bounds
end if
end for

end for

Bacterial Foraging Optimization has found a niche in the literature. In [20], several applications are collected. BFO has been used to train a wavelet-based neural network [21]. Pattern recognition (for example, [22] and [23]) is a problem that has been studied as well using various variants of BFO. Dynamic resource allocation for Multiple Input, Multiple Output (MIMO) experimental platforms was tackled in [24]. In terms of estimation theory, BFO was applied in [25] to improve the quality of solutions for the Extended Kalman Filter (EKF) with a focus on applications to solve the simultaneous localization and mapping (SLAM) problem. The famous scheduling problem was tackled by numerous authors, including, for example [26]. Furthermore, in control theory [27] used a modified BFO algorithm to optimize coefficients of a

proportional-integral (PI) controller. It is readily apparent that BFO has found many dynamic uses in the applied sciences and engineering.

2.3 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Covariance Matrix Adaption Evolutionary Strategy was originally set out in [28] and is a matrix-based algorithm designed to have a computational cost of $O(n^2)$. Hansen's paper introducing the algorithm relies heavily on [29]. The Generating Set Adaptation (GSA) was proposed in [29] as the first evolutionary strategy invariant to coordinate system rotations and that learned a particular problem's scaling. GSA did not include the formal covariance matrix in the algorithm. Then, using this foundation, Hansen in [28] introduced the $(1, \lambda)$-ES (where $\mu = 1$) CMA-ES algorithm. The new algorithm used a covariance matrix with great success, yielding an algorithm that, while still heuristic, has a more concrete theoretical basis in statistical analysis than many other algorithms found in evolutionary computation.

After declarations and initializations, the details of CMA-ES can be written in around 20 lines of MATLAB code. Hansen provides source code on his website [30], so rather than going into extensive theoretical detail, an outline of the algorithm is presented based on [28]. Given the search space dimension n and the iteration step k, it is possible to define the following five variables. Define $m_k \in \mathbb{R}^n$ to be the distribution mean and current favored solution. Further define and set $\sigma_k > 0$ to be the step size. Then, notate C_k to be a symmetric positive definite $n\ x\ n$ covariance matrix, initialized with $C_0 = I$, the identity matrix. Finally, define $p_\sigma \in \mathbb{R}^n$ and $p_c \in \mathbb{R}^n$ to be two evolution paths initially defined as the zero vector. Select λ, the number of samples per iteration (also known as the population size). The algorithm is dependent on the selection of λ, making the algorithm quasi-parameter free for the user. This is an advantage of

CMA-ES over other algorithms that sometimes have many parameters that need to be defined through trial and error.

The candidate solutions are $x_i \sim \mathcal{N}(m_k, \sigma_k^2 C_K)$. It is then possible to denote the candidate solutions as

$$\{x_{i:\lambda} \mid i = i \dots \lambda\} = \{x_i \mid i = 1 \dots \lambda\} \tag{2.3}$$

and

$$f(x_{1:\lambda}) \leq \cdots \leq f\left(x_{\mu:\lambda}\right) \leq f\left(x_{\mu+1:\lambda}\right) \dots \tag{2.4}$$

The new mean is then easily computed via

$$m_{k+1} = \sum_{i=1}^{\mu} w_i x_{i:\lambda} = m_k + \sum_{i=1}^{\mu} w_i (x_{i:\lambda} - m_k) \tag{2.5}$$

where the recombination weights w_i, where $w_1 \geq w_2 \geq \cdots \geq w_\mu > 0$ all sum to one. Typically, one would select $\mu \leq \frac{\lambda}{2}$ and with weights $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \approx \frac{\lambda}{4}$. The step size is updated using the cumulative step-size adaptation (CSA). Define $c_\sigma^{-1} \approx \frac{n}{3} > 1$ to be the backwards time horizon for the evolution path p_σ. Further define $\mu_w = \left(\sum_{i=1}^{\mu} w_i^2\right)^{-1}$ to be the variance effective selection mass, noting that by virtue of how w_i is defined that $1 \leq \mu_w \leq \mu$. A unique symmetric square root of the inverse of C_k is $C_k^{-1/2} = \left(\sqrt{C_k}\right)^{-1} = \sqrt{C_k^{-1}}$. Additionally, define d_σ to be a damping parameter, usually close to one; note that as $d_\sigma \to \infty$ or $c_\sigma \to 0$ that the step-size does not change. It is then possible to begin the iteration. It is now possible to write equations that update p_σ and σ_{k+1}, namely

$$p_{\sigma+1} = (1 - c_\sigma) p_\sigma + \sqrt{1 - (1 - c_\sigma)^2} \sqrt{\mu_w} C_k^{-1/2} \frac{m_{k+1} - m_k}{\sigma_k} \tag{2.6}$$

and

$$\sigma_{k+1} = \sigma_k \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{E\|\mathcal{N}(0,I)\|} - 1\right)\right) \tag{2.7}$$

Now, σ_k will be increased if and only if $\|p_\sigma\|$ is larger than the expected value

$$E\|\mathcal{N}(0,I)\| = \sqrt{2}\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)} \approx \sqrt{n}\left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right) \tag{2.8}$$

Otherwise, if the expectation is smaller, it will be decreased. Finally, the covariance matrix is updated. Define again $c_c^{-1} \approx \frac{n}{4} > 1$ to be the backwards time horizon for the evolution path p_c. Let $\alpha \approx 1.5$. The indicator function $\mathbf{1}_{[0,\sigma\sqrt{n}]}(\|p_\sigma\|)$ evaluates to one if and only if (which normally is the case) $\|p_\sigma\| \leq \alpha\sqrt{n}$. Additionally, $c_1 \approx \frac{2}{n^2}$ is the learning rate for the rank one update of the covariance matrix. Furthermore, $c_\mu \approx \frac{\mu_w}{n^2}$ is the learning rate for the rank μ update of the covariance matrix; this may not exceed $1 - c_1$. Piecing together these definitions, now define the value for $c_s = (1 - \mathbf{1}_{[0,\sigma\sqrt{n}]}(\|p\|)^2)c_1c_c(2 - c_c)$. It is now finally possible to write

$$p_{c+1} = (1 - c_c)p_c + \mathbf{1}_{[0,\sigma\sqrt{n}]}(\|p_\sigma\|)\sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w}\frac{m_{k+1} - m_k}{\sigma_k} \tag{2.9}$$

and

$$C_{K+1} = \left(1 - c_1 - c_\mu + c_s\right)C_k + c_1p_cp_c^T + c_\mu\sum_{i=1}^{\mu} w_i\frac{x_{i:\lambda} - m_k}{\sigma_k}\left(\frac{x_{i:\lambda} - m_k}{\sigma_k}\right)^T \tag{2.10}$$

The covariance matrix update tends to increase the likelihood for p_c and for $\frac{x_{i:\lambda} - m_k}{\sigma_k}$ to be sampled from $\mathcal{N} \sim (0, C_{k+1})$. This concludes the update step. The algorithm then continues to iterate until terminated by a termination procedure determined by the user.

CMA-ES has been used often in the literature, so only a few highlights are presented here. In [31], CMA-ES was employed to aid in forensic identification. More applicable to the discipline of aerospace engineering would be the application in [32] where the authors worked on turbulent friction drag reduction. Optimization is a pressing matter in the design of control

systems; many papers (for example, see [32]) use CMA-ES to optimize feedback controllers. In [34], CMA-ES was employed to optimize noise sensor selection. CMA-ES has also been used in relation to other areas in evolutionary computation; in [35], CMA-ES was use to optimize neural field models. Another aerospace application that has been explored is the optimization of compressor blades [36]. The problem of lens design was tackled in [37]. This particular evolutionary strategy has also made its way into the medical field, yielding papers such as [38] where CMA-ES was employed towards the heuristic design of cancer chemotherapies. Clearly, CMA-ES has become a popular algorithm of choice for many researchers, in part because it is easy to use and implement.

Chapter 3

Optimal Two-Impulse Transfers Between Circular Orbits

The Hohmann transfer is a famous problem in astrodynamics. The Hohmann transfer and its proof are discussed in detail in Section 3.1. For this purposes of this thesis, the Hohmann transfer provides an opportunity to explore the performance of the algorithms to a problem that has a well-known solution. Section 3.2 presents the problem statement used in Chapter 4 and Section 3.3 discusses the numerical implementation of that problem statement. Ideally, the results from the algorithm should approach the limit of the Hohmann results.

3.1 Hohmann Transfer background

Walter Hohmann in 1925 first postulated that an orbital transfer between two circular co-planar orbits under the standard assumptions would be the minimum energy transfer between those orbits [39]. The mechanics of this transfer would involve a first impulse to remove the spacecraft from its initial circular orbit onto an elliptical transfer orbit. Then, once the spacecraft has traveled from the one apse point of the transfer ellipse to the other apse point, a second impulse is applied to circularize the spacecraft into its new orbit. The fundamental assumption is that the thrusts are impulsive: in other words that the orbital radii do not change (in theory, at all – in practice, appreciably) over the duration of the thrust.

Define for convenience $\beta = \frac{R_2}{R_1}$, where R_1 represents the radius of the inner circular orbit and R_2 represents the radius of the outer circular orbit. The Hohmann transfer is only optimal over a specific range of values. As presented in [40], the following table holds true.

Table 3.1. Range of β Values and Effect on Optimality

Range of β	Conclusion
$\beta \in (0, 11.94)$	The Hohmann transfer is optimal
$\beta \in (11.94, 15.58)$	Sometimes the Hohmann transfer is optimal; need to test for optimality
$\beta \in (15.58, \infty)$	The bi-elliptic transfer can be superior

The finite end points of the ranges are known as the critical limits of the transfer problem. As noted above, when $\beta \in (11.94, 15.58)$, an additional test is needed to determine the optimality of the Hohmann transfer. Escobal in [41] presents a test and provides equations to test for the conditions needed.

Various proofs of the Hohmann transfer have been given over the years. A proof using the calculus of variations are given in [42] and in [43]. Some more creative approaches and alternative proofs are given in [44] (using Green's Theorem), [45] (using graphical construction) and [46] (using Lagrange multipliers). More recent work includes a proof by Palmore [47] using the techniques of elementary calculus. While the Hohmann transfer itself is relatively easy to calculate, one must calculate all possible alternative transfers in order to complete the proof – not an easy task in most cases.

The proof presented below was originally inspired by Prussing [48] and makes comparisons to the proof presented by Palmore. In Palmore's proof, every possible transfer orbit is listed using nonlinear functions of the semi-latus rectum p and eccentricity e. The boundaries of the relevant region of the (p, e) plane that then contain the feasible regions for the possible transfers contain difficult curves. It then follows that the gradient of the characteristic velocity of the orbital maneuver with respect to these two variables can be calculated to determine the minimum Δv, which corresponds to minimization of the propellant needed. Instead, make two changes to Palmore's approach. The first adaptation is the use the variables of the semi-latus

rectum and eccentricity *themselves*, instead of non-linear functions thereof in Palmore's presentation. The second adaptation is to use only the partial derivative of the characteristic velocity with respect to eccentricity.

In order for one to find the boundaries of the feasible region, one needs to recall the orbit equation, a polar equation for orbital radius as a function of true anomaly f

$$r = \frac{p}{1 + e\cos f} \tag{3.1}$$

It is simple to note that the semi-latus rectum (also known as the parameter by some authors) is defined from analytic geometry to be

$$p = a(1 - e^2) \tag{3.2}$$

where a represents the semi-major axis of the orbit. One can readily observe that the semi-latus rectum is a function of the size and shape of a given orbit. A feasible region means that the transfer orbit will intersect both R_1 and R_2. Assume that $\beta > 1$ holds true. An orbit can be labeled as infeasible if either the periapsis lies outside the smaller terminal radius or the apoapsis lies within the larger terminal orbit. Then, from Equation (3.1), the following two conditions need to be satisfied

$$r_p = \frac{p}{1 + e} \leq r_1 \tag{3.3}$$

and

$$r_a = \frac{p}{1 - e} \geq r_2 \tag{3.4}$$

Equations (3.3) and (3.4) can be rather easily rewritten to find the semi-latus rectum. Carrying this out reveals

$$p \leq r_1(1 + e) \tag{3.5}$$

and

$$p \geq r_2(1 - e) \tag{3.6}$$

Every possible transfer orbit can be written as functions of semi-latus rectum and eccentricity and takes its place as a point in the (p, e) plane. The inequalities given by Equations (3.5) and (3.6) define the region of all of the possible transfer orbits. It then follows that the boundaries of these regions will be straight lines.

The law of cosines provides the formula for the magnitude of the velocity change to enter a circular orbit or depart from a circular orbit at radius $r = r_k$

$$(\Delta v)^2 = v^2 + v_c^2 - 2v_c v_\theta \tag{3.7}$$

where $k = 1, 2$, v_c is the circular orbital speed and v_θ is the component of the velocity vector that is normal or perpendicular to the radius. Note the equations for the conservation of angular momentum

$$v_\theta = \frac{h}{r} = \frac{\sqrt{\mu p}}{r} \tag{3.8}$$

where h is the specific angular momentum of the orbit (often simply called angular momentum) and μ is the standard gravitational parameter. Further, the velocity can be found from the vis-viva integral

$$v^2 = \mu\left(\frac{2}{r} - \frac{1}{a}\right) \tag{3.9}$$

For historical reasons, it should be noted that integral in this sense has the archaic meaning of "constant" that predates the more modern definition of the word seen in elementary calculus. It can then be easily shown that

$$v_c^2 = \frac{\mu}{r} \tag{3.10}$$

Through substitution, Equation (3.9) can be rewritten as

$$v^2 = \mu\left(\frac{2}{r} + \frac{e^2 - 1}{p}\right) \tag{3.11}$$

The characteristic velocity is

$$\Delta v_T = \Delta v_1 + \Delta v_2 \tag{3.12}$$

where, to ensure consistent notation, Δv_1 occurs at a radius of R_1 and Δv_2 occurs at a radius of R_2. Upon calculating the partial derivate with respect to eccentricity one finds that

$$\frac{\partial \Delta v_T}{\partial e} = \frac{\partial \Delta v_1}{\partial e} + \frac{\partial \Delta v_2}{\partial e} \tag{3.13}$$

By way of Equations (3.7) and (3.11), it is then possible to write

$$\Delta v_k \frac{\partial \Delta v_k}{\partial e} = v_k \frac{\partial v_k}{\partial e} = \frac{e\mu}{p} \tag{3.14}$$

It then immediately follows that

$$\frac{\partial \Delta v_T}{\partial e} = \frac{e\mu}{p}\left(\frac{1}{\Delta v_1} + \frac{1}{\Delta v_2}\right) > 0 \tag{3.15}$$

It is possible to note that since the partial derivative is positive then any point on the interior of the feasible region can be lowered by lowering the value of eccentricity while holding semi-latus rectum constant. Define r_p to be the radius at periapsis and r_a to be the radius at apoapsis. This implies that minimum characteristics velocity will lie on the intersection of $r_a = R_2$ and $r_p = R_1$. What needs to be shown is that if the characteristic velocity is constrained to the boundary its derivative with respect to eccentricity will remain positive. In other words, the optimal solution lies at the minimum value of eccentricity on the boundary.

The most feasible way to do this is to recognize that the expression for the characteristic velocity is constrained to the boundary. Now it is possible to substitute for the value of the semi-latus rectum along each portion of the boundary, namely with $p = R_2(1 - e)$ on the left and $p = R_1(1 + e)$ on the right. It is then possible to write the characteristic velocity as a function of eccentricity. Define $\Delta\tilde{v}$ to be the velocity change constrained to the boundary. Equation (3.7) then takes the form

$$(\Delta\tilde{v})^2 = \mu\left[\frac{2}{r} + \frac{e^2 - 1}{R_2(1 - e)}\right] + \frac{\mu}{r} - 2\sqrt{\frac{\mu}{r}}\frac{\sqrt{\mu R_2(1 - e)}}{r} \tag{3.16}$$

By differentiating with respect to eccentricity and noting that $B > 1$ and $e \in (0, 1)$ for an ellipse, one finds that

$$2\Delta\tilde{v}_1 \frac{d\Delta\tilde{v}_1}{de} = \frac{\mu}{r}\left[\frac{\beta^{\frac{3}{2}}}{\sqrt{1-e}} - 1\right] > 0 \tag{3.17}$$

and

$$2\Delta\tilde{v}_2 \frac{d\Delta\tilde{v}_2}{de} = \frac{\mu}{r_2}\left[\frac{1}{\sqrt{1-e}} - 1\right] > 0 \tag{3.18}$$

Similar to what was seen with Equation (3.15), on the left portion of the boundary $\frac{d\Delta\tilde{v}_T}{de} > 0$. Likewise, on the right portion of the boundary

$$(\Delta\tilde{v})^2 = \mu\left[\frac{2}{r} + \frac{e^2-1}{R_1(1+e)}\right] + \frac{\mu}{r} - 2\sqrt{\frac{\mu}{r}}\frac{\sqrt{\mu R_1(1+e)}}{r} \tag{3.19}$$

Differentiating yields

$$2\Delta\tilde{v}_1 \frac{d\Delta\tilde{v}_1}{de} = \frac{\mu}{r_1}\left[1 - \frac{1}{\sqrt{1+e}}\right] > 0 \tag{3.20}$$

and

$$2\Delta\tilde{v}_2 \frac{d\Delta\tilde{v}_2}{de} = \frac{\mu}{r}\left[1 - \frac{1}{\beta^{\frac{3}{2}}\sqrt{1+e}}\right] > 0 \tag{3.21}$$

Following the similar trend, $\frac{d\Delta\tilde{v}_T}{de} > 0$ holds on the right side of the boundary as well. The derivative of $\Delta\tilde{v}_T$ with respect to the eccentricity is positive for all points on the boundary. It then reasonably follows that the optimal solution lies at the point of minimum eccentricity on the boundary – in other words, with the Hohmann transfer.

3.2 Problem Statement

Now that the theory is understood, it is possible to turn one's attention to numerical demonstrations. As originally presented in [14], this problem involves finding the optimal locations, directions and magnitudes of two impulses. We seek to minimize the fitness (also known as the objective or cost) function

$$J = \Delta v_1 + \Delta v_2 \tag{3.22}$$

where the magnitudes of the two impulsive changes in velocity are represented by Δv_1 and Δv_2, respectively. Define μ_B as the standard gravitational parameter of the attracting body. Then, the initial conditions at time t_0^- are

$$v_r(t_0^-) = 0 \tag{3.23}$$

$$v_\theta(t_0^-) = \sqrt{\frac{\mu_B}{R_1}} \tag{3.24}$$

and

$$r(t_0^-) = R_1 \tag{3.25}$$

where v_r and v_θ denote the radial and the horizontal component of the velocity vector $\overline{v}$, respectively, and r is defined as the radius of the orbit. The terminal conditions at time t_f^+ are then captured by

$$v_r\left(t_f^+\right) = 0 \tag{3.26}$$

$$v_\theta\left(t_f^+\right) = \sqrt{\frac{\mu_B}{R_2}} \tag{3.27}$$

and

$$r\left(t_f^+\right) = R_2 \tag{3.28}$$

The velocity components v_r and v_θ change to the following expressions after the first impulse

$$v_r(t_0^+) = v_r(t_0^-) + \Delta v_1 \sin \delta_1 \tag{3.29}$$

and

$$v_\theta(t_0^+) = v_\theta(t_0^-) + \Delta v_1 \cos \delta_1 \tag{3.30}$$

Note, of course, that $r(t_0^+) = r(t_0^-) = R_1$ due to the impulsive approximation; in other words, the radius does not change appreciably over the duration of the orbital maneuver. One is able to define the time of the second impulse as time t_f. Then, during the time period between impulses, $t \in [t_0^+, t_f^-]$, the trajectory is modeled as Keplerian. Note that a is the classical orbital element that represents the semi-major axis and that e is the classical orbital element that represents the eccentricity of the Keplerian arc of the spacecraft's trajectory Due to this construction, these equations

$$a = \frac{\mu_B r(t_0^+)}{2\mu_B - r(t_0^+)\left(v_r^2(t_0^+) + v_\theta^2(t_0^+)\right)} \tag{3.31}$$

and

$$e = \sqrt{1 - \frac{r^2(t_0^+)v_\theta^2(t_0^+)}{\mu_B a}} \tag{3.32}$$

hold true. Recalling the definition of the semi-latus rectum from Equation (3.2), one can then write an expression for true anomaly through the forms

$$\cos\left(f(t_0^+)\right) = \frac{v_\theta(t_0^+)}{e}\sqrt{\frac{p}{\mu_B}} - \frac{1}{e} \tag{3.33}$$

and

$$\sin\left(f(t_0^+)\right) = \frac{v_r(t_0^+)}{e}\sqrt{\frac{p}{\mu_B}} \tag{3.34}$$

where $f(t_0^-)$ denotes the true anomaly at time t_0^+. One can easily reason that the coasting arc must terminate when the radius becomes R_2. By extension, the true anomaly $f(t_0^+)$ immediately before the second impulse is given by

$$f\left(t_f^-\right) = \cos^{-1}\left(\frac{p - R_2}{R_2 e}\right) \tag{3.35}$$

At time t_f^-, v_r and v_θ can be written as

$$v_r\left(t_f^-\right) = \sqrt{\frac{\mu_B}{p}}\, e \sin\left(f\left(t_f^-\right)\right) \tag{3.36}$$

and

$$v_\theta\left(t_f^-\right) = \sqrt{\frac{\mu_B}{p}}\left[1 + e\cos\left(f\left(t_f^-\right)\right)\right] \tag{3.37}$$

Equations (3.26), (3.27) and (3.28), the terminal conditions, instruct that

$$v_r\left(t_f^+\right) = v_r\left(t_f^-\right) + \Delta v_2 \sin\delta_2 = 0 \tag{3.38}$$

and

$$v_\theta\left(t_f^+\right) = v_\theta\left(t_f^-\right) + \Delta v_2 \cos\delta_2 = \sqrt{\frac{\mu_B}{R_2}} \tag{3.39}$$

These equations can be used to find the magnitude and direction of the second impulse

$$\Delta v_2 = \sqrt{v_r^2\left(t_f^-\right) + \left(\sqrt{\frac{\mu_B}{R_2}} - v_\theta\left(t_f^-\right)\right)} \tag{3.40}$$

$$\cos\delta_2 = \frac{\sqrt{\frac{\mu_B}{R_2}} - v_\theta\left(t_f^-\right)}{\Delta v_2} \tag{3.41}$$

and

$$\sin\delta_2 = -\frac{v_r\left(t_f^-\right)}{\Delta v_2} \tag{3.42}$$

It then reasonably follows that these two inequality constraints must be satisfied by any feasible transfer trajectory

$$a > 0 \tag{3.43}$$

and

$$a(1+e) \geq R_2 \tag{3.44}$$

In other words, the transfer orbit must be elliptic, and the apoapsis radius must be greater than or equal to the final radius R_2. As defined, this problem is non-linear programming problem subject to the inequality constraints of Equations (3.43) and (3.44). The two unknown parameters are Δv_1 and δ_1, the impulse magnitude and direction, respectively.

3.3 Numerical Implementation

Each candidate solution contains the two unknown parameters

$$\chi = [\Delta v_1 \ \delta_1]^T \tag{3.45}$$

Canonical units are employed in order to use a normalized set of units. The initial radius represents the distance unit (DU) and the time unit (TU) is defined using the standard gravitational parameter such that $\mu_B = 1$ DU^3/TU^2. Now, for PSO and BFO the optimal values of the two unknown parameters are bounded by

$$0 \ \frac{DU}{TU} \leq \Delta v_1 \leq 1 \frac{DU}{TU} \tag{3.46}$$

and

$$-\pi \leq \delta_1 \leq \pi \tag{3.47}$$

The candidate solution may not violate Equations (3.43) or (3.44), the inequality constraints. CMA-ES does not support parameter bounding *de jure* by its very nature, so a penalty was added to the objective function to simulate the parameter bounding *de facto* if these boundaries were violated as described further below.

Two penalty cases were examined. The first (hereinafter known as the fixed penalty case) involved applying fixed penalties of magnitude 100 if the inequality constraints were violated.

The second (hereinafter the varying penalty case) involved applying varying penalties of $100|a|$ if Equation (3.43) was violated and $100(a(1+e)-R_2)^2$ if Equation (3.44) was violated. Since CMA-ES does not support parameter bounding, for the fixed penalty case the additional penalty was 100 for each violation of a parameter's bound. For the varying penalty case, the additional penalty was $100|\chi_i|$ for each violation of a parameter's bound.

To guard against the occasional divergence scenario, if an algorithm returned a result with error greater than 25%, the algorithm was restarted and the initial results were thrown out. PSO was run with 30 particles for 500 iterations using cognitive and social scaling parameters of 1.49445 and an inertial weight of $\frac{1+rand(n)}{2}$, where $rand(n)$ denotes a uniformly distributed random number $\in (0,1)$. BFO was run 30 bacteria that each swam for 100 steps for 500 generations a swim step size of 0.000001. The probably of elimination and dispersal used for BFO was 0.25. CMA-ES was initialized with a step size of 0.1. The foregoing was then implemented numerically and results are presented as Chapter 4.

Chapter 4

Results and Discussion

The choice of programming language one may use is often problem specific. Many researchers have had great success with programming various algorithms in Evolutionary Computation in C++ or Java, especially when parallelization has been a concern. For this experiment, it was deemed that MATLAB would be sufficient for this task since MATLAB has excellent support of the extensive matrix operations needed to carry out CMA-ES. The numerical simulation described in Chapter 3 was easy to implement in MATLAB, the results of which are presented in Section 4.1 and analyzed in Section 4.2

4.1 Numerical Results

Since PSO was the algorithm used in [14], it made sense to begin with that algorithm here. Using the settings listed in Section 3.3, the algorithm was run once using fixed penalties.

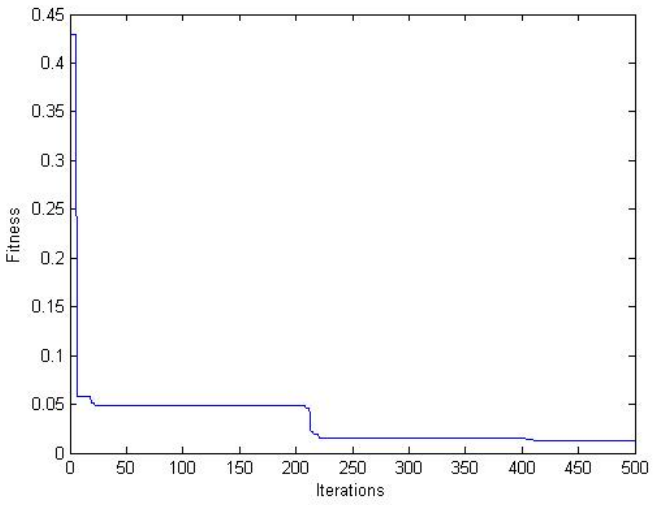

Figure 4.1. Fitness vs. Iterations of β = 2 for a Single Run of PSO using Fixed Penalties

Fitness is defined throughout as the calculated objective function minus the known result, *i.e.* the Hohmann results. As can be seen in Figure 4.1, the fitness improves quickly over a relatively small number of iterations. It is then possible to look at the performance metrics.

Table 4.1. Performance Metrics of β = 2 for a Single Run of PSO using Fixed Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.140	0.284	0.297	4.55%

Table 4.1 presents the performance metrics for the run in Figure 4.1 Note that due to rounding, percent errors may not line up exactly with the values in the table. The MATLAB function *cputime* was used to calculate the central processing unit (CPU) time needed to run the algorithm. The processing was done using a quad core Intel i5-2400 processor. As can be readily seen from Table 4.1, the algorithm is fast – carrying out the sub-second function evaluations rather quickly. The percent error calculated is less than 5%, which is not bad considering no gradients were used to carry out the optimization. Next, the algorithm was run 10 times to see if changing $\beta = \frac{R_2}{R_1}$ would affect the algorithm's performance. The results are presented below in Figure 4.2

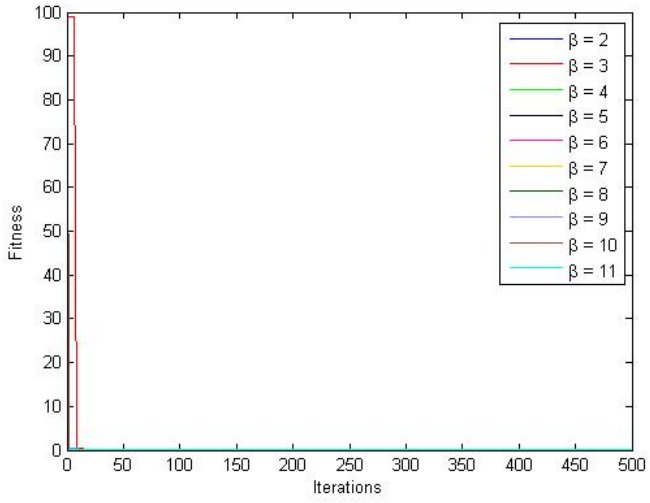

Figure 4.2. Fitness vs. Iterations of Various β for Single Runs of PSO using Fixed Penalties

As can be seen in Figure 4.2, the fitness improves so rapidly that the resolution of the graph doesn't immediately provide useful information. Figure 4.3 below provides a zoom of the fitness history for a selected range of iterations.

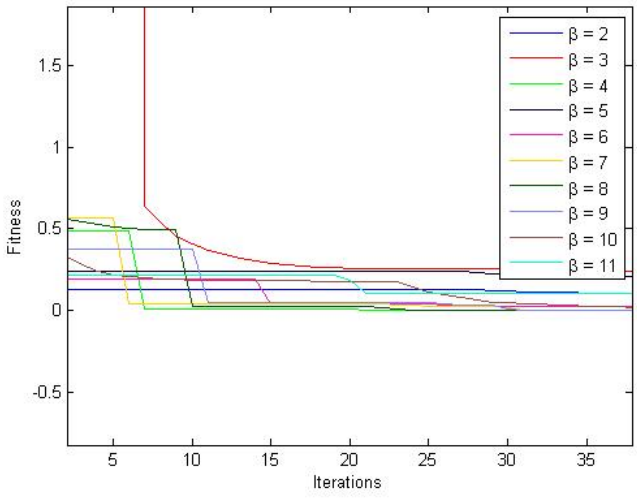

Figure 4.3. Detailed View of Fitness vs. Iterations of Various β for Single Runs of PSO using Fixed Penalties

It is readily apparent from Figure 4.3 that the algorithm performs well across a range of β. The upper limit of β is selected, of course, with consideration to Table 3.1 knowing the limitation of the Hohmann formulation. As before, it is possible to look at performance metrics for the results that had the best fitness. These are tabulated below in Table 4.2.

Table 4.2. Performance Metrics of Various β for a Single Run of PSO with Fixed Penalties

$\boldsymbol{\beta}$	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.140	0.284	0.293	2.99%
3	0.125	0.394	0.397	0.74%
4	0.140	0.449	0.449	0.13%
5	0.109	0.480	0.481	0.24%
6	0.109	0.499	0.499	0.00%
7	0.125	0.512	0.514	0.43%
8	0.109	0.520	0.520	0.00%
9	0.125	0.526	0.526	0.00%
10	0.094	0.530	0.532	0.50%
11	0.125	0.532	0.534	0.23%

Upon examination of Table 4.2, it is difficult to discern a pattern as β varies. The reasonable conclusion, then, is that the choice of β did not affect the performance of the algorithm for the fixed penalty case. Now, running the code 1000 times gives one the opportunity to assess the statistics of the algorithm's performance. Figure 4.4 shows a box plot of the percent error of PSO using fixed penalties. Figure 4.5 shows a box plot of the CPU time for the same 1000 runs.

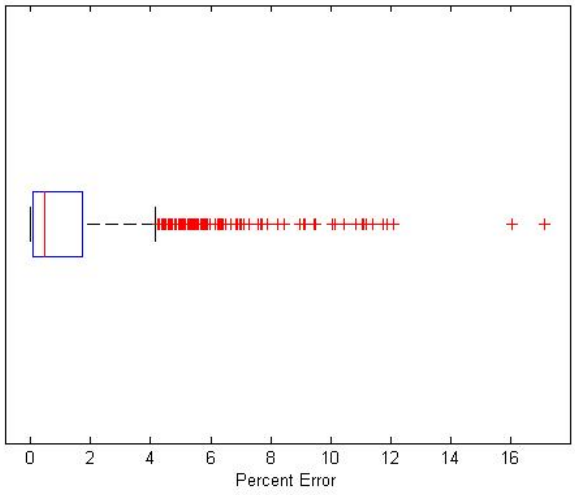

Figure 4.4. Box Plot of β = 2 for 1000 Runs of PSO with Fixed Penalties for Percent Error

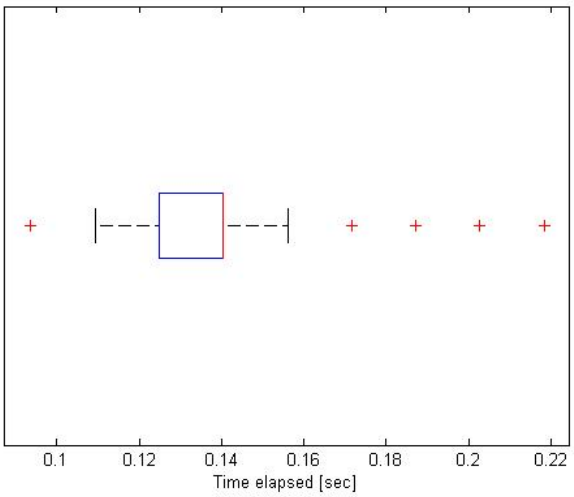

Figure 4.5. Box Plot of β = 2 for 1000 Runs of PSO with Fixed Penalties for Time Elapsed

All box plots in this thesis use the following notation: the box contains values between the 25th and 75th percentiles, the central mark in the box indicates the median value, the vertical bars outside of the box indicate the extremes and the "+" signs denote outliers, if present. Figure 4.4 indicates that there is skewing in the error sample, with the occasional high outlier. This is reasonable – not every initial guess is going to return a great result with a fixed number of function evaluations (NFE). Figure 4.5 shows a fairly small range of CPU times, which justifies not calculating the median CPU time in the performance metrics. These performance metrics are found below in Table 4.3.

Table 4.3. Performance Metrics of β = 2 for 1000 Runs of PSO using Fixed Penalties

β	Mean CPU Time [secs]	Theoretical Total Δv [TU/DU]	Mean Calculated Total Δv [TU/DU]	Median Calculated Total Δv [TU/DU]	Mean Percent Error	Median Percent Error
2	0.137905	0.284	0.288	0.286	1.40%	0.48%

The results for 1000 runs are excellent. Once again, the skewing is visible in comparing the mean and median percent error, though they are not terribly far apart. This suggests once again that the occasional bad initial guess can propagate through the algorithm and cause slower optimization when the algorithm gets hung up on local extrema. Now it is possible to examine the varying penalty case. Figure 4.6 shows a graph of fitness against iteration for a single run of PSO, this time with the varying penalties. Figure 4.7 shows a detailed view of Figure 4.6.

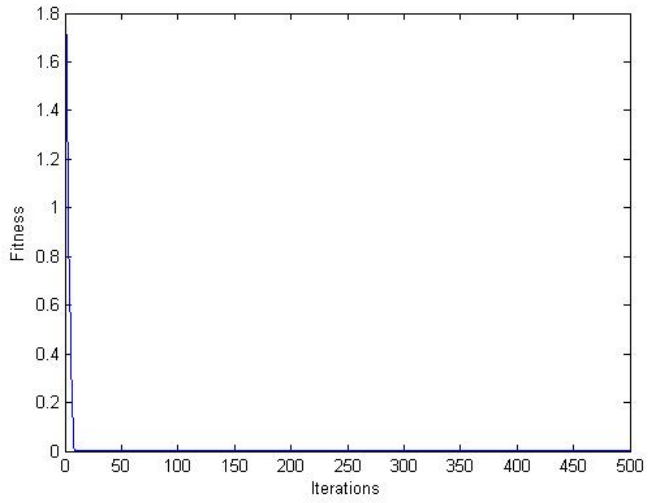

Figure 4.6. Fitness vs. Iterations of β = 2 for a Single Run of PSO using Varying Penalties

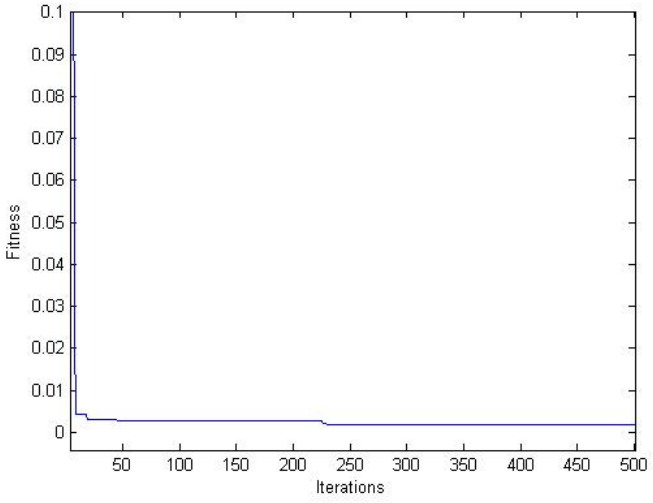

Figure 4.7. Detailed View of Fitness vs. Iterations of β = 2 for a Single Run of PSO using Varying Penalties

On casual observation, one could observe that the algorithm seems to settle towards optimum more quickly than in the fixed penalty case though one must be careful not to draw too much information from a single run of the algorithm. The performance metrics from this run are shown in Table 4.4.

Table 4.4. Performance Metrics of β = 2 for a Single Run of PSO using Varying Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.125	0.284	0.286	0.59%

The percent error seen in Table 4.4 for this run is quite good and the CPU time is faster than had been seen in the single run of the fixed penalty case. Now, letting β vary again, it is possible to present Figure 4.8.

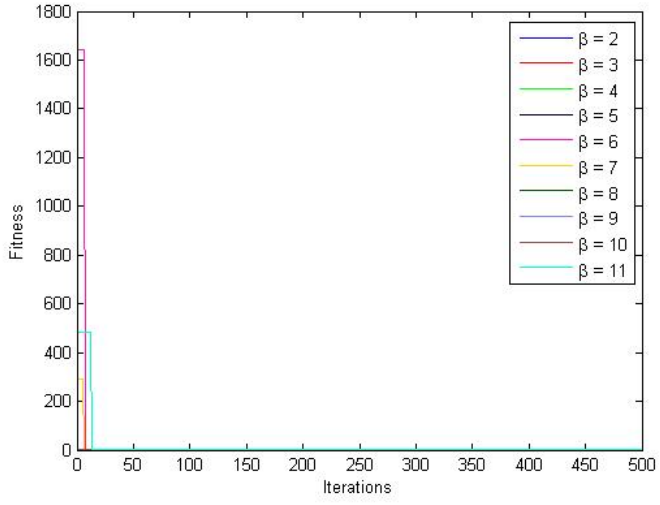

Figure 4.8. Fitness vs. Iterations of Various β for Single Runs of PSO using Varying Penalties

Similar performance as before was achieved, though the resolution is once again a little difficult for the casual observer to make out. Since the algorithm converges rather quickly, a detailed view of this plot is presented as Figure 4.9.

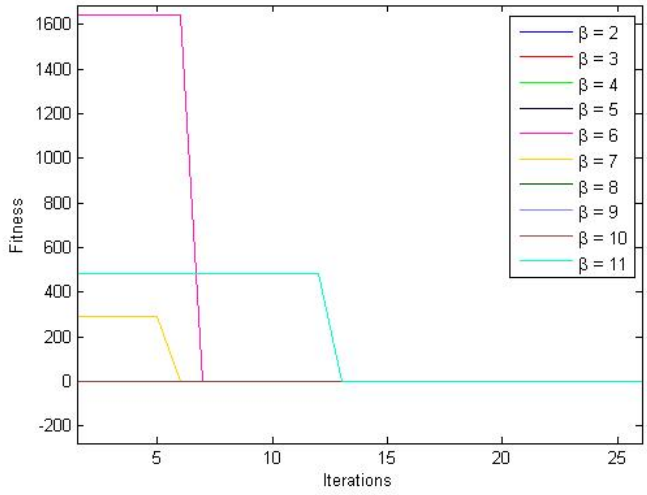

Figure 4.9. Detailed View of Fitness vs. Iterations of Various β for Single Runs of PSO using Varying Penalties

It should be noted that the lines for fitness are so close together that they are on top of each other. The performance metrics were tabulated once again and are presented below in Table 4.5.

Table 4.5. Performance Metrics of Various β for a Single Run of PSO with Varying Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.109	0.284	0.286	0.55%
3	0.094	0.394	0.398	1.07%
4	0.094	0.449	0.458	2.17%
5	0.094	0.480	0.480	0.06%
6	0.109	0.499	0.512	2.48%
7	0.156	0.512	0.560	9.46%
8	0.109	0.520	0.548	5.43%
9	0.094	0.526	0.550	4.53%
10	0.094	0.530	0.545	2.83%
11	0.109	0.532	0.533	0.03%

Once again, the trend appears to be that the choice of β has minimal impact on the accuracy of the solution. This speaks well to the robustness of PSO. Now, the algorithm was run 1000 times. The boxplots for percent error is presented in Figure 4.10.

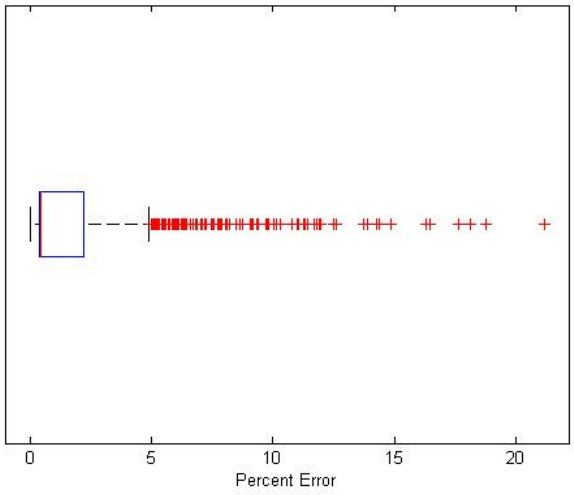

Figure 4.10. Box Plot of β = 2 for 1000 Runs of PSO with Varying Penalties for Percent Error

Additionally, the box plot for CPU time is found below in Figure 4.11.

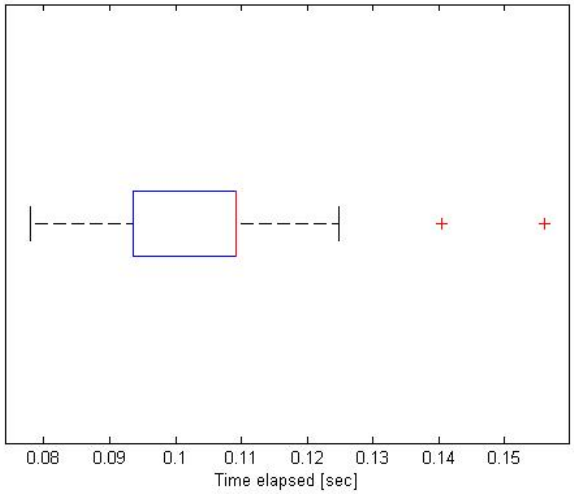

Figure 4.11. Box Plot of β = 2 for 1000 Runs of PSO with Varying Penalties for Time Elapsed

The mean percent error is lower to the first quartile than it was in the fixed penalty case and the mean run time has improved. This improvement is perhaps most obvious when viewing Table 4.6, in which the performance metrics are tabulated.

Table 4.6. Performance Metrics of β = 2 for 1000 Runs of PSO using Varying Penalties

$\boldsymbol{\beta}$	Mean CPU Time [secs]	Theoretical Total Δv [TU/DU]	Mean Calculated Total Δv [TU/DU]	Median Calculated Total Δv [TU/DU]	Mean Percent Error	Median Percent Error
2	0.107	0.284	0.289	0.285	1.78%	0.43%

Since the CPU time decreased, one's initial reaction may be to expect the error to improve markedly as well. This reaction, however, is not substantiated by Table 4.6; while the median error does decrease the mean error increases, though neither difference is especially large.

Now that the performance of PSO has been characterized, it is possible to look at BFO. Continuing with the same pattern as before, BFO was run once with fixed penalties. The fitness plot is presented below in Figure 4.12.

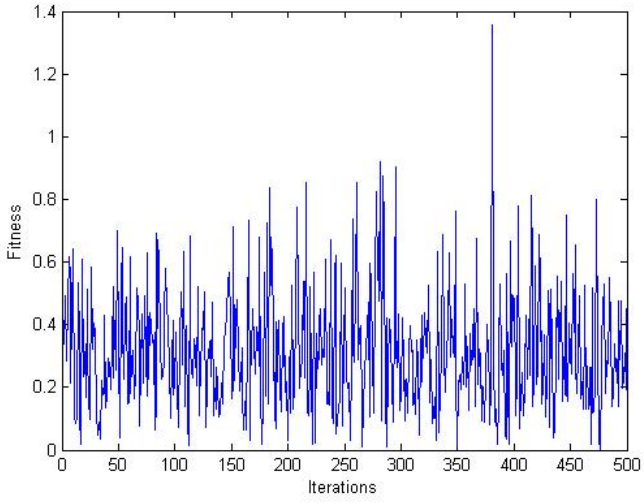

Figure 4.12. Fitness vs. Iterations of β = 2 for a Single Run of BFO using Fixed Penalties

The plot has stronger spikes to it because each bacterium is allows to swim, though the fitness seems to center not far from the known solution (*i.e.* fitness that is zero). The performance metrics are tabulated in Table 4.7.

Table 4.7. Performance Metrics of β = 2 for a Single Run of BFO using Fixed Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.624	0.284	0.287	0.76%

While the percent error seems to be along the lines of that seen with PSO, it is obvious that the CPU time has gone up; this is likely a function of the need to simulate the 100 swim steps of each bacterium. Now, vary β and it is possible to develop Figures 4.13 and 4.14.

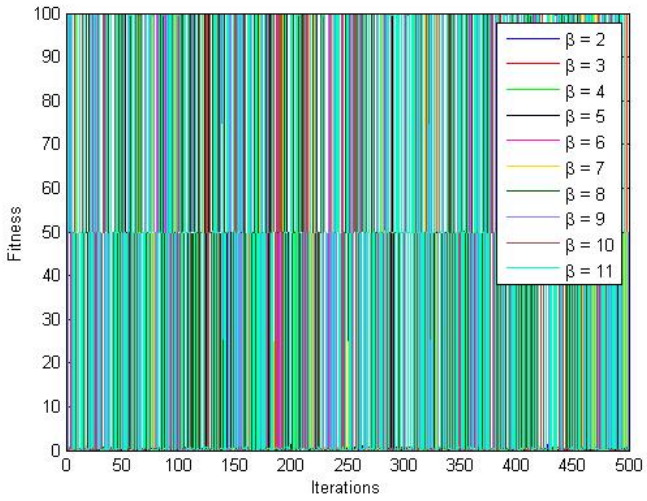

Figure 4.13. Fitness vs. Iterations of Various β for Single Runs of BFO using Fixed Penalties

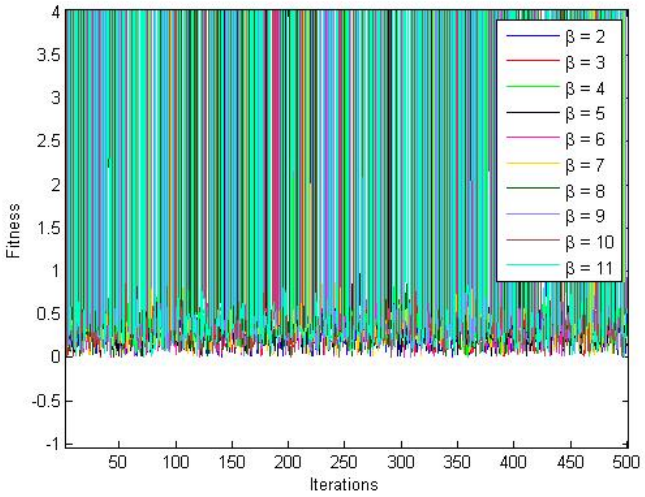

Figure 4.14. Detailed View of Fitness vs. Iterations of Various β for Single Runs of BFO using Fixed Penalties

The spikes in fitness degrade the appearance of the plots. Figure 4.14 offers a closer look on what is occurring in Figure 4.13. The performance metrics are then presented in Table 4.8.

Table 4.8. Performance Metrics of Various β for a Single Run of BFO with Fixed Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.702	0.284	0.293	3.09%
3	0.608	0.394	0.399	1.37%
4	0.671	0.449	0.460	2.58%
5	0.640	0.480	0.493	2.61%
6	0.671	0.499	0.505	1.15%
7	0.686	0.512	0.526	2.83%
8	0.655	0.520	0.529	1.75%
9	0.686	0.526	0.531	0.97%
10	0.702	0.530	0.534	0.82%
11	0.624	0.532	0.541	1.68%

The algorithm appears to perform no better on large values of β than it does for small values of β and the results are fairly good overall. Now, the code was run 1000 times again. The box plot of percent error follows below as Figure 4.15.

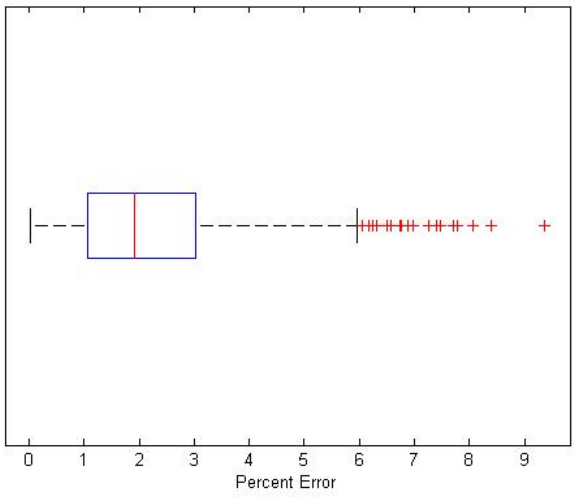

Figure 4.15. Box Plot of β = 2 for 1000 Runs of BFO with Fixed Penalties for Percent Error

Along the same lines, Figure 4.16 is the box plot of the CPU time for the fixed penalty case of BFO.

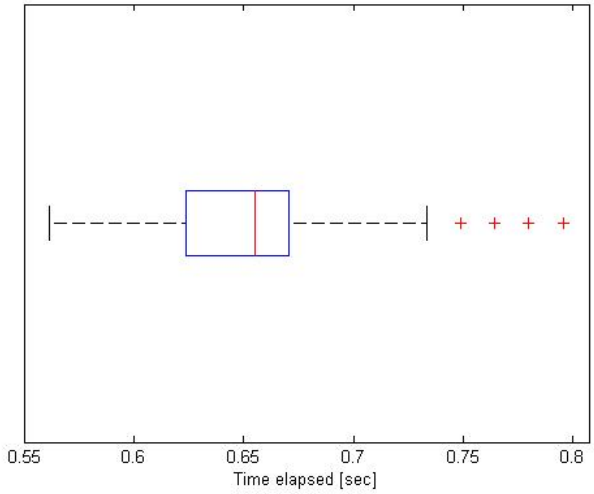

Figure 4.16. Box Plot of β = 2 for 1000 Runs of BFO with Fixed Penalties for Time Elapsed

The box plot in Figure 4.15 suggests a fairly small range of errors for the 1000 runs. Likewise, the spread of CPU times is fairly small as well. The performance metrics are tabulated below as Table 4.9.

Table 4.9. Performance Metrics of β = 2 for 1000 Runs of BFO using Fixed Penalties

$\boldsymbol{\beta}$	Mean CPU Time [secs]	Theoretical Total Δv [TU/DU]	Mean Calculated Total Δv [TU/DU]	Median Calculated Total Δv [TU/DU]	Mean Percent Error	Median Percent Error
2	0.655	0.284	0.291	0.290	2.19%	1.91%

The mean and median errors are fairly close together, which matches the impression given by Figure 4.15. The fairly large mean time for this application stands out as noteworthy. Now, the varying penalty case can be implemented. The single run iteration history follows in Figure 4.17.

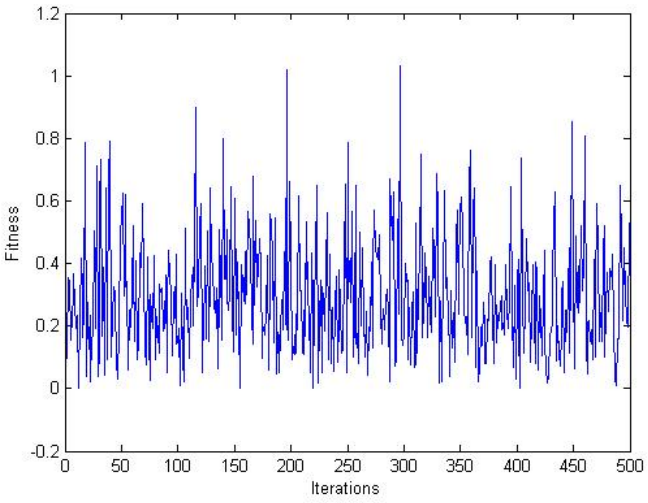

Figure 4.17. Fitness vs. Iterations of β = 2 for a Single Run of BFO using Varying Penalties

Once again, similar spiky performance is seen as each bacterium swims. The performance metric for this run follow in Table 4.10.

Table 4.10. Performance Metrics of β = 2 for a Single Run of BFO using Varying Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.749	0.284	0.284	0.15%

The CPU time appears to have increased again, though one must continue to be careful not to read too much from a single run of the code. The eye is drawn towards the remarkably low percent error – this suggests paying attention to the error for this case. Figure 4.17 on the next page shows the iteration history for various values of β. This plot is much cleaner than Figure 4.13, and therefore needs no detailed view. It appears that fitness does occasionally spike very high, but that these occurrences are fairly rare.

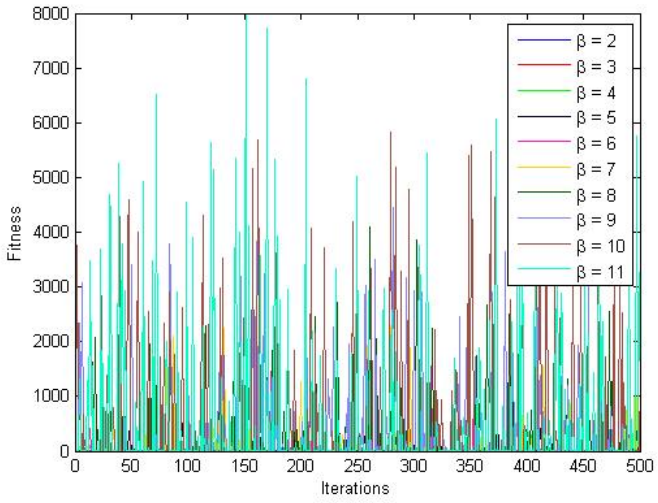

Figure 4.18. Fitness vs. Iterations of Various β for Single Runs of BFO using Varying Penalties

The performance metrics from these varying cases are tabulated and appear below in Table 4.11. On the whole, the CPU time appear high again, though the percent errors achieved appear to be excellent.

Table 4.11. Performance Metrics of Various β for a Single Run of BFO with Varying Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.718	0.284	0.285	0.09%
3	0.702	0.394	0.405	2.72%
4	0.671	0.449	0.453	0.87%
5	0.702	0.480	0.483	0.68%
6	0.718	0.499	0.500	0.07%
7	0.686	0.512	0.515	0.71%
8	0.780	0.520	0.521	0.16%
9	0.671	0.526	0.526	0.04%
10	0.718	0.530	0.549	3.62%
11	0.733	0.532	0.538	1.01%

The code is then run 1000 times with an eye towards viewing the statistical trends once again. Figure 4.19 the box plot for the error, while Figure 4.20 shows the box plot of the CPU time.

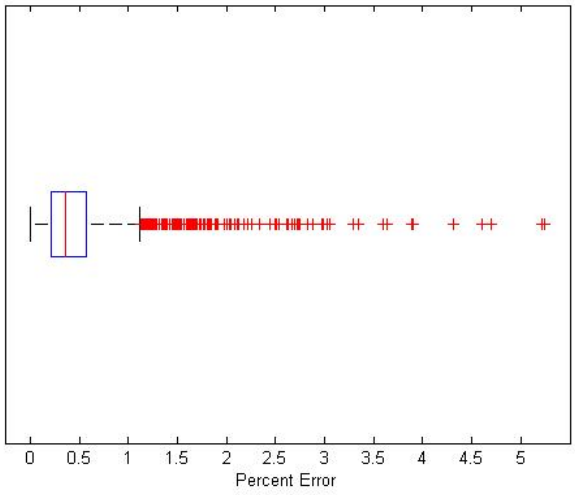

Figure 4.19. Box Plot of β = 2 for 1000 Runs of BFO with Varying Penalties for Percent Error

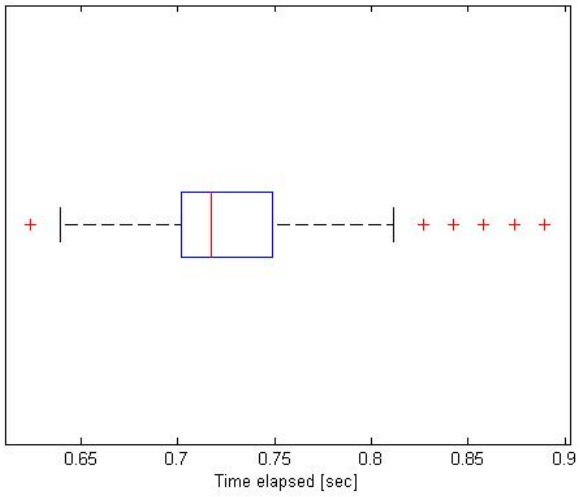

Figure 4.20. Box Plot of β = 2 for 1000 Runs of BFO with Varying Penalties for Time Elapsed

There appears to be a fair number of outliers for the percent error as seen in Figure 4.18, though the error itself looks to be fairly good. This is likely a function of the relatively small interquartile range. The trend of fairly large run times that had been seen in single runs is confirmed by the box plot in Figure 4.20.

Table 4.12. Performance Metrics of β = 2 for 1000 Runs of BFO using Varying Penalties

$\boldsymbol{\beta}$	Mean CPU Time [secs]	Theoretical Total Δv [TU/DU]	Mean Calculated Total Δv [TU/DU]	Median Calculated Total Δv [TU/DU]	Mean Percent Error	Median Percent Error
2	0.727	0.284	0.285	0.285	0.27%	0.36%

The performance metrics for the varying penalty case are tabulated above in Table 4.12. The mean CPU time is around three quarters of a second, which is remarkably high considering each function evaluation is sub-second in duration. While the run time is up, the percent errors are down and the closeness between the mean and median percent errors suggests there is not as much skewing of this data set.

The last algorithm examined is CMA-ES. Continuing with the pattern of single runs, a single run was performed with the modified fixed penalty case developed in Section 3.3.

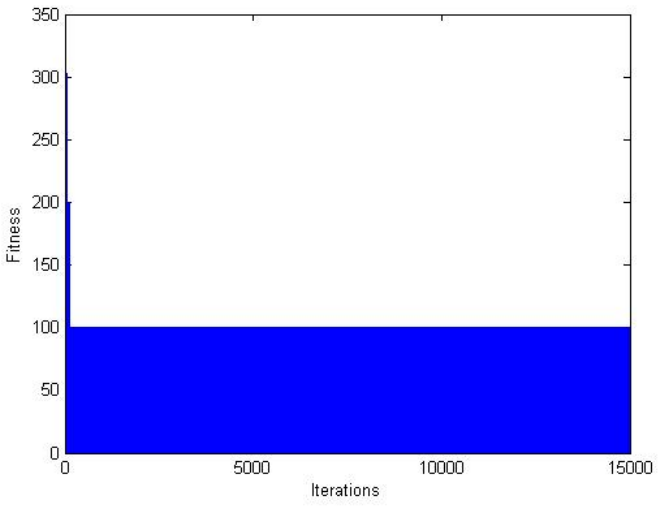

Figure 4.21. Fitness vs. Iterations of β = 2 for a Single Run of CMA-ES using Fixed Penalties

This plot, at first glance, appears to be nonsense. However, upon zooming in on the plot, it because readily apparent that the fitness value is oscillating from iteration to iteration. Performance metrics follow in Table 4.13.

Table 4.13. Performance Metrics of β = 2 for a Single Run of CMA-ES using Fixed Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.546	0.284	0.284	0.00%

The percent error, as evidenced in Table 4.13, achieved in this single run is surprising, especially when viewed in light of the CPU time.

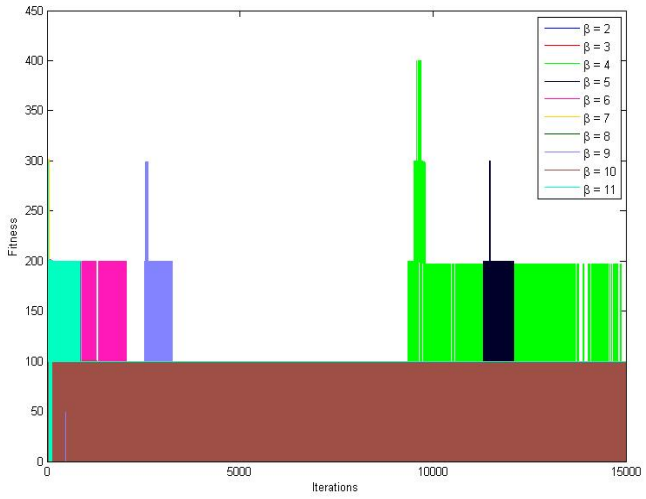

Figure 4.22. Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Fixed Penalties

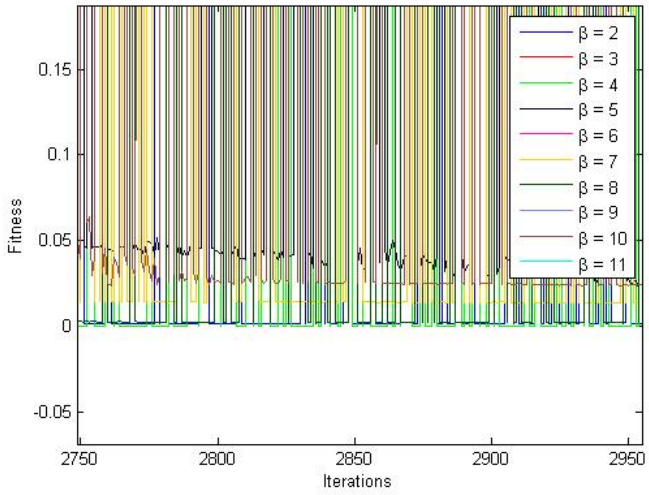

Figure 4.23. Detailed View of Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Fixed Penalties

The iteration history is shown in Figures 4.22 and 4.23. Once again, the resolution of the global plot is fairly poor in Figure 4.22, so a zoomed view is presented in Figure 4.23. The first plot is clarified once the reader notices that the colors of the various lines are blurring to produce colors such as brown and magenta. The second plot is the more interesting of the two and really displays the oscillation of the fitness values. Table 4.14 presents the performance metrics from these runs.

Table 4.14. Performance Metrics of Various β for a Single Run of CMA-ES with Fixed Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.546	0.284	0.284	0.00%
3	0.562	0.394	0.406	3.11%
4	0.546	0.449	0.449	0.00%
5	0.562	0.480	0.480	0.00%
6	0.562	0.499	0.507	1.49%
7	0.562	0.512	0.512	0.00%
8	0.546	0.520	0.520	0.00%
9	0.593	0.526	0.560	6.45%
10	0.577	0.530	0.530	0.00%
11	0.577	0.532	0.553	3.84%

The performance metrics once again are good; the consistency in CPU run time stands out.

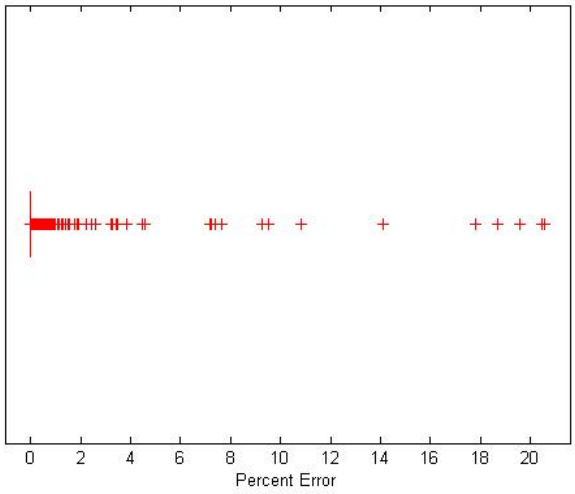

Figure 4.24. Box Plot of β = 2 for 1000 Runs of CMA-ES with Fixed Penalties for Percent Error

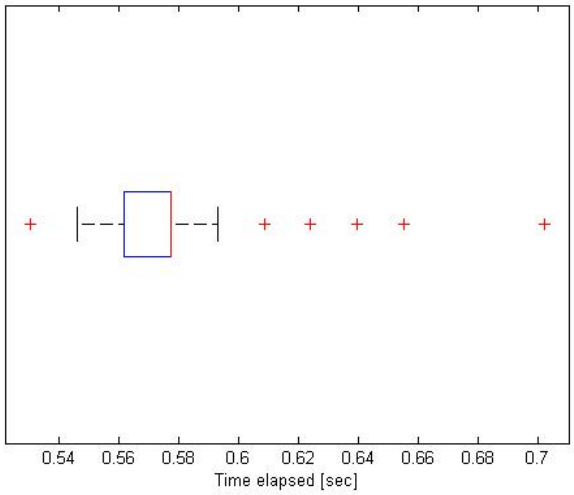

Figure 4.25. Box Plot of β = 2 for 1000 Runs of CMA-ES with Fixed Penalties for Time Elapsed

Figures 4.24 and 4.25 show the box plots for percent error and time elapsed, respectively. In Figure 4.24, one can see that the interquartile range collapses because most of the errors are approaching 0%. The time elapsed range is fairly small for the interquartile range; there are not many outliers. The performance metrics follow below in Table 4.15.

Table 4.15. Performance Metrics of β = 2 for 1000 Runs of CMA-ES using Fixed Penalties

$\boldsymbol{\beta}$	Mean CPU Time [secs]	Theoretical Total Δv [TU/DU]	Mean Calculated Total Δv [TU/DU]	Median Calculated Total Δv [TU/DU]	Mean Percent Error	Median Percent Error
2	0.572	0.284	0.285	0.284	0.26%	0.00%

The median percent error is striking, and the mean percent error is fantastic as well. The mean CPU time matches the consistently seen in previous runs of the algorithm. The question that leads from this data is if this trend will hold up for the varying penalty case. A single run of that case follows in Figure 4.26. A detailed view then follows in Figure 4.27.

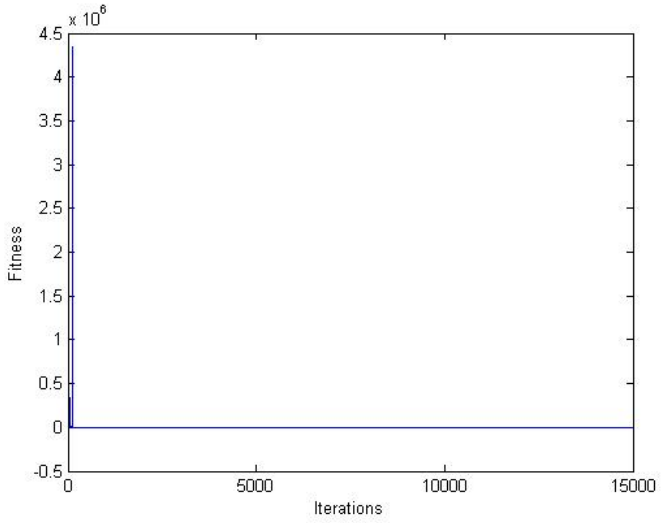

Figure 4.26. Fitness vs. Iterations of β = 2 for a Single Run of CMA-ES using Varying Penalties

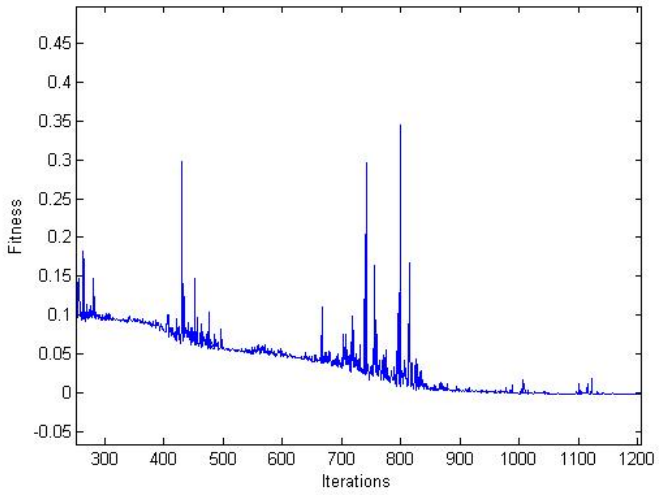

Figure 4.27. Detailed View of Fitness vs. Iterations of β = 2 for a Single Run of CMA-ES using Varying Penalties

The fitness improves exponentially over just a few iterations of the algorithm. This is desirable behavior because longer evaluation CPU times cost more computationally and, by extension, financially. The performance metrics for this case are then tabulated and presented in Table 4.16.

Table 4.16. Performance Metrics of β = 2 for a Single Run of CMA-ES using Varying Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.593	0.284	0.283	0.42%

The CPU time seen in Table 4.16 is consistent with what had been seen before and remains respectable. The percent error also compares well to runs of the other algorithms.

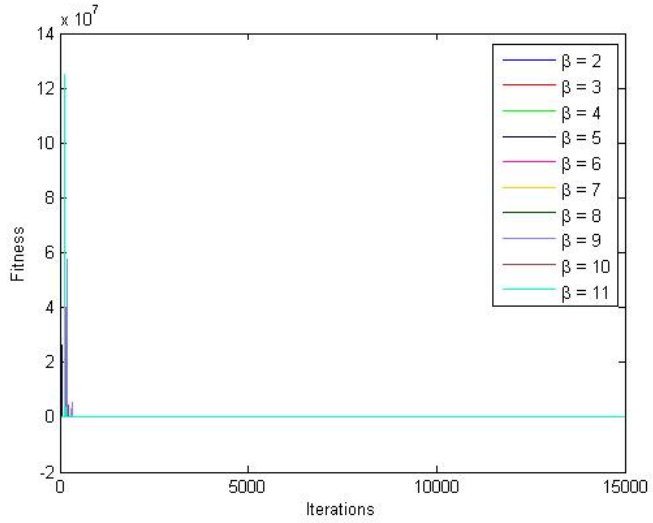

Figure 4.28. Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Varying Penalties

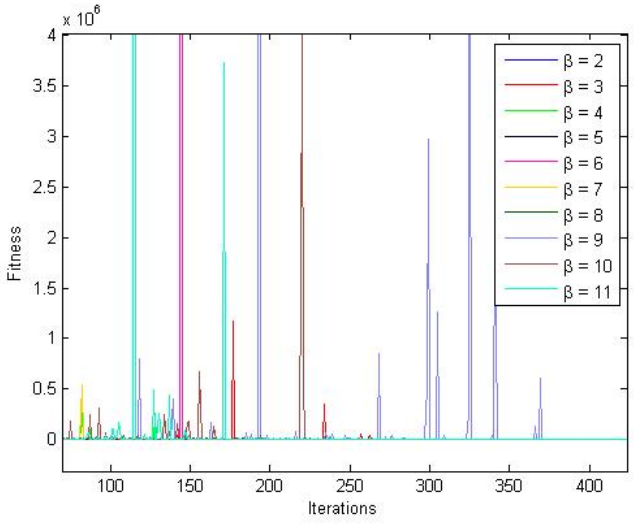

Figure 4.29. Detailed View of Fitness vs. Iterations of Various β for Single Runs of CMA-ES using Varying Penalties

The previous two figures represent 10 runs for various values of β. The exponential decay continues to be seen, as are the fitness spikes. Interestingly, however, the spikes occur less frequently for the varying penalty case than they do for the fixed penalty case. The spikes, admittedly, are higher, though the algorithm seems to do a better job of realizing it has come across a poor guess and it quickly moves away from it. The performance metrics are presented below in Figure 4.17. The percent error for these 10 runs appears to be rather small for the most part, though the percent error for the $\beta = 2$ case doesn't seem to follow the trend.

Table 4.17. Performance Metrics of Various β for a Single Run of CMA-ES with Varying Penalties

β	CPU Time [secs]	Theoretical Total Δv [TU/DU]	Calculated Total Δv [TU/DU]	Percent Error
2	0.562	0.284	0.283	0.43%
3	0.608	0.394	0.394	0.08%
4	0.577	0.449	0.449	0.03%
5	0.577	0.480	0.480	0.01%
6	0.562	0.499	0.499	0.01%
7	0.562	0.512	0.512	0.00%
8	0.593	0.520	0.520	0.00%
9	0.593	0.526	0.526	0.00%
10	0.577	0.530	0.530	0.00%
11	0.562	0.532	0.532	0.00%

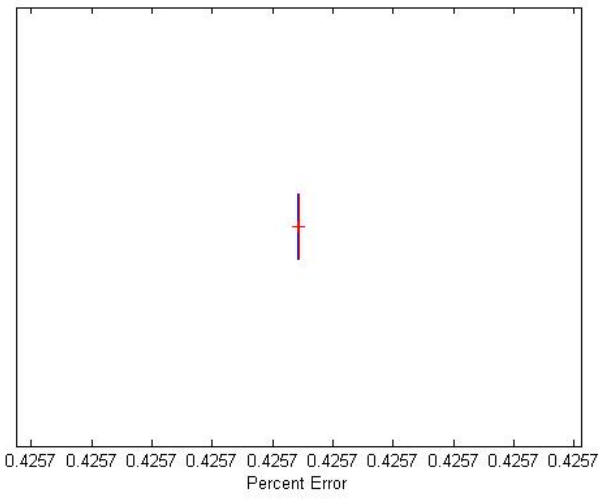

Figure 4.30. Box Plot of β = 2 for 1000 Runs of CMA-ES with Varying Penalties for Percent Error

The trend is then further explored for 1000 iterations. The box plot of percent error is presented in Figure 4.30. The percent error for the 1000 runs is striking. It appears that the error strongly centers on 0.425%, and exhibits behavior that appears to mimic a constant bias. The box plot of the CPU times follows in Figure 4.31.

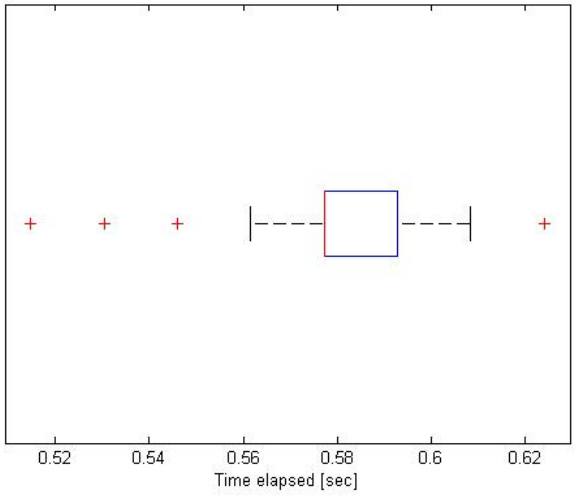

Figure 4.31. Box Plot of β = 2 for 1000 Runs of CMA-ES with Varying Penalties for Time Elapsed

The median appears to collapse onto the first quartile of data, with lower CPU times appearing to be more likely as outliers than high CPU times. The detailed performance metrics are presented below in Table 4.18.

Table 4.18. Performance Metrics of β = 2 for 1000 Runs of CMA-ES using Varying Penalties

$\boldsymbol{\beta}$	Mean CPU Time [secs]	Theoretical Total Δv [TU/DU]	Mean Calculated Total Δv [TU/DU]	Median Calculated Total Δv [TU/DU]	Mean Percent Error	Median Percent Error
2	0.582	0.284	0.283	0.283	0.43%	0.43%

What is most striking about CMA-ES with the varying penalties is its consistency. In addition to the CPU times being fairly uniform, the mean and median error are one and the same. This

suggests the complete absence of skewing, which is confirmed from the box plot of the percent error.

4.2 Discussion

PSO and BFO are more alike to each other than either of them is similar to CMA-ES. This is understandable since both are Evolutionary Algorithms designed beyond the traditional genetic algorithm approach. PSO models swarming behavior; BFO models bacterial foraging. CMA-ES is a somewhat more rigorous (while still heuristic) example of an Evolutionary Strategy. The differences between Evolutionary Algorithms and Evolutionary Strategies are largely historical in nature, the result of two varying viewpoints in how algorithms are designed in the Evolutionary Computation community.

In terms of CPU time, PSO performed strongly in this study due to its ease of implementation and relatively low computational overhead. CMA-ES involved more computations due to its matrix-based design, but performed faster in MATLAB than BFO did. This revelation is not surprising since MATLAB is optimized for matrix operations and is notoriously slow at handling loops and other control structures. BFO appears to lag behind the other algorithms because of the necessary simulation of the swimming behavior of each individual bacterium.

Only CMA-ES required the restarts described in Section 3.3. For the fixed penalty case, the single run performed required a restart. The 10 runs for the varying β cases together needed 42 restarts. This trend continues with the 1000 runs as well for fixed penalties. It is obvious that CMA-ES had trouble traversing the sudden penalties imposed by the constraint violation. For the varying penalty case, no restarts were needed for the single run, and only 2 restarts were needed to do the 10 runs for the varying β cases. This problem of the Hohmann-like transfer was a good

test subject because the solution is well known, and can be proven analytically as was done in Chapter 2. The question this poses, then, is how would a user know to restart CMA-ES if the algorithm were to diverge? Despite the algorithm's pseudo-rigorous nature, no one has to date been able to demonstrate convergence or divergence criteria for the algorithm and the fact that the algorithm is quasi-parameter free does not leave a lot of parameters to experiment with to try to improve convergence.

On the subject of parameters, BFO's performance was likely influenced by the choice of search parameters. The parameters for all of the algorithms were chosen experientially because they appeared to give good results. The problem with BFO is that there are a larger number of parameters that need to be set. This criticism is one often attached to genetic algorithms as well. If one had the computational resources, one could use Evolutionary Computation to optimize the algorithm to optimize this problem. This smacks very strongly of Zero's dichotomy paradox in that to the inexperienced user the task will then appear to be hopeless. This naturally greatly increases the time and skill needed to achieve good results. Another approach that is meta-heuristic could be to create an outer loop that adjusts the parameters and repeatedly solves the problem until both the optimum parameters and the problem solution are found.

A common criticism of Evolutionary Computation is that the algorithms are computationally expensive. This is true. Evolutionary should not be applied to problems when it is possible or feasible to calculate gradients. Gradient-based optimization is still generally preferable. However, it is not always possible to calculate a gradient – the gradient may not exist, or it may be difficult to find as is often the case in science and engineering problems. Based on this, Evolutionary Computation has a niche in the world of optimization and Evolutionary Computation will continue to be used to solve difficult problems.

Chapter 5

Conclusions and Future Work

As noted by many authors, Evolutionary Computation has exploded in popularity in the applied science and engineering fields. Many Evolutionary Algorithms and Evolutionary Strategies are tremendously easy to use and implement. Performance of these algorithms is often a function of problem complexity and difficulty. The problem studied in this thesis has the advantage of being well-known and it has a well-studied solution; due to this, it is a good choice as a test problem.

5.1 Lessons Learned

There are many lessons that can be learned from this thesis. One of the most obvious results is that algorithm superiority is generally a myth. It is impossible to demonstrate rigorously the performance of an algorithm for all kinds of problems, though many authors have tried to demonstrate it for a variety of individual cases. Some algorithms will perform better on some problems and this will often vary as a function of problem difficulty and complexity. Programming language choice also makes a difference. CMA-ES has a lot of overhead with matrix operations, yet it performed faster in terms of CPU time than BFO because MATLAB, the language of choice for this study, is optimized for matrix operation and is well known to lag for control structures such as loops. This observed behavior is likely to vary as a function of problem size. The role of constraints and penalties play a critical role in the implementation of these algorithms, yet their role is often inconsistent and hard to pin down. Some algorithms like CMA-

ES don't natively support constraints and so a work-around has to be developed to enforce that the algorithm should not search outside of the constraints.

5.2 Future Work

Future work in Evolutionary Computation as applied to space trajectory optimization is bright. This thesis focused mainly on the standard versions of PSO, BFO and CMA-ES. Many variants exist in the literature that tackle some of the flaws that these algorithms inherently have. Another problem of strong interest is the finite-thrust case (the so called "burn-coast-burn" trajectory), which is more difficult to simulate because of the need for numeric integration. Outside of single-objective optimization, many authors are working on multi-objective algorithms that are able to optimize multiple variables simultaneously (*e.g.* cost *and* time). Regardless of which approach will be tackled next, Evolutionary Computation truly remains a ripe area of research in the field of astrodynamics.

References

[1] Bäck, T., Hammel, U. and Schwefel, H-P., "Evolutionary Computation: Comments on the History and Current State," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, April 1997, pp. 3-17.

[2] Bremermann, H.J., "Optimization through evolution and recombination", in *Self-Organizing Systems*, edited by M. C. Yovits *et al*., Spartan, Washington, DC, 1962.

[3] Friedberg, R.M., "A Learning Machine: Part I," *IBM Journal*, Vol. 2, No. 1, January 1958, pp. 2-13.

[4] Friedberg, R.M., "A Learning Machine: Part II," *IBM Journal*, Vol. 3, No. 7, July 1959, pp. 282-287.

[5] Box, G.E.P., "Evolutionary Operation: A Method for Increasing Industrial Productivity," *Applied Statistics*, Vol. VI, No. 2, 1957, pp. 81-101.

[6] Holland, J.H., "Outline for a Logical Theory of Adaptive Systems," *Journal of the Association of Computer Machinery*, Vol. 9, Issue 3, July 1962, pp. 297-314.

[7] Rechenberg, I., "Cybernetic Solution Path of an Experimental Problem," Royal Aircraft Establishment, Library Translation No. 1122, Farnborough, Hants., U.K., August 1965.

[8] Schwefel, H-P., "Experimentelle Optimierung einer Zweiphasendiise Teil I", AEG Research Institute Project MHD-Staustrahlrohr 11034/68 Technical Report 35.

[9] Fogel, L.J, "Autonomous Automata," *Industrial Research Magazine*, Vol. 4, No. 2, February 1962, pp. 14-19.

[10] Moore, G.E., "Cramming More Components Onto Integrated Circuits," *Electronics*, Volume 38, Number 8, April 1965, pp. 114-117.

[11] Kennedy, J., and Eberhart, R. "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks*, Vol. 4, Institute of Electrical and Electronics Engineers, Western Australia, 1995, pp. 1942-1948.

[12] Poli, R. "Analysis of the Publications on the Applications of Particle Swarm Optimization," *Journal of Artificial Evolution and Applications*, Vol. 2008, Article ID 685175, 10 pages, 2008.

[13] Shi, Y. and R. C. Eberhart, "A Modified Particle Swarm Optimizer," *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '99)*, Institute of Electrical and Electronics Engineers, Piscataway, NY, 1999, pp. 69-73.

[14] Pontani, M. and Conway, B.A., "Particle Swarm Optimization Applied to Space Trajectories," *Journal of Guidance, Control and Dynamics*, Vol. 33, No. 5, 2010, pp. 1429-1441.

[15] Passino, K.M., "Biomimicry of Bacterial Foraging for Distributed Optimization and Control," *IEEE Control System Magazine*, Vol. 22, Issue 3, pp. 52-66, June 2002.

[16] Bentley, R., and Meganathan, R., "Biosynthesis of Vitamin K (Menaquinone) in Bacteria," *Microbiological Reviews*, Vol. 46, No. 3, pp. 241-280, September 1982.

[17] "*E. Coli (Escherichia coli),*" Centers for Disease Control and Prevention. [www.cdc.gov/ecoli/index.html. Accessed 7/1/13.]

[18] Melton, R.G., "Hybrid Methods for Determining Time-Optimal, Constrained Spacecraft Reorientation Maneuvers," *Acta Astronautica*, http://dx.doi.org/10.1016/j.actaastro.2013.05.007, in press.

[19] Chen, H., Zhu, Y., and Hu., K., "Adaptive Bacterial Foraging Optimization," *Abstract and Applied Analysis*, Vol. 2011, 2011, 27 pages.

[20] Das, S., Biswas, A., Dasgupta, S. and Abraham, A., "Bacterial Foraging Optimizion Algorithm: Theoretical Foundations, Analysis, and Applications," *Foundations of*

Computational Intelligence, Vol. 3, Edited by Abraham, A., Hassanien, A-E., Siarry, P. and Engelbrecht, A., Springer, 2009, pp. 23-55.

[21] Ulagammai, L., Vankatesh, P., Kannan, P.S., Padhy, N.P., "Application of Bacterial Foraging Technique Trained and Artificial and Wavelet Neural Networks in Load Forcasting," *Neurocomputing*, Vol. 70, Issues 16-18, October 2007, pp. 2659-2667, http://dx.doi.org/10.1016/j.neucom.2006.05.020.

[22] Dasgupta, S., Biswas, A., Das, S., and Abraham, A., "Automatic Circle Detection on Images with an Adaptive Bacterial Foraging Algorithm," *2008 Genetic and Evolutionary Computation Conference (GECCO 2008)*, ACM Press, New York, 2008, pp. 1695-1696.

[23] Acharya, D.P., Panda, G., Mishra, S. and Lakhshmi, Y.V.S., "Bacteria Foraging Based Independent Component Analysis," *International Conference on Computational Intelligence and Multimedia Applications*, IEEE Press, Los Alamitos, December 2007, pp. 527-531.

[24] Munoz, M.A., Lopez, J.A., Caicedo, E., "Bacterial Foraging Optimization for Dynamical Resource Allocation in a Multizone Temperature Experimentation Platform," *Analysis and Design of Intelligent Systems using Soft Computing Techniques, Vol. 41*, Edited by Melin, P., Castillo, O., Ramírez, E.G., Kacprzyk, J., and Pedryck, W., Springer, 2007, pp. 427-435.

[25] Chatterjee, A. and Matsuno, F., "Bacertial Foraging Techniques for Solving EKF-Based SLAM Problems, *Proceedings of the International Control Conference (Control 2006)*, Glasgow, U.K., August 30-September 01, 2006.

[26] Wu, C., Zhang, N., Jiang, J., Yang, J. and Liang, Y., "Improved Bacterial Foraging Algorithms and Their Applications to Job Shop Scheduling Problems," *Adaptive and Natural Computing Algorithms: 8th International Conference, ICANNGA 2007*,

Proceedings, Part I, Edited by Beliczynski, B., Dzielinski, A., Iwanowski, M. and Ribeiro, B., Warsaw, Poland, April 11-14, 2007, pp. 562-569.

[27] Mishra, S. and Bhende, C.N., "Bacterial Foraging Technique-Based Optimized Active Power Filter for Load Compensation," *IEEE Transactions on Power Delivery*, Vol. 22, No. 1, January 2007, pp. 457-465.

[28] Hansen, N.A., Ostermeier, A., "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, 1996, pp. 312-317.

[29] Hansen, N.A., Ostermeier, A., and Gawelczyk, A., "On the Adaptation of Arbitrary Normal Mutation Distributions in Evolutionary Strategies: The Generating Set Adaptation," *Proceedings of the Sixth International Conference on Genetic Algorithms*, Edited by Eshelman, Pittsburgh, PA, 1995, pp. 57-64.

[30] Hanson, N.A., personal website. [https://www.lri.fr/~hansen/. Accessed 7/1/2013.]

[31] Ibañez, O., Ballerini, L., Cordón, O., Damas, S., and Santamaría, "An Experimental Study on the Applicability of Evolutionary Algorithms to Craniofacial Superimposition in Forensic Identification," *Information Sciences*, Vol. 179, 2009, pp. 3998-4029.

[32] Fukagata, K., Kern, S., Chatelain, P., Koumoutsakos, P., and Kasagi, N., "Evolutionary Optimization of an Anisotropic Compliant Surface for Turbulent Friction Drag Reduction," *Journal of Turbulence*, Vol. 9, No. 35, 2008, pp. 1-17.

[33] Hansen, N., Niederberger, A.S.P., Guzzella, L., and Koumoutsakos, P., "Evolutionary Optimization of Feedback Controllers for Thermoacoustic Instability," *IUTAM Symposium on Flow Control and MEMS, Proceedings held at the Royal Geographical Society, 19-22 September 2006*, Edited by Morrison, J., Birch, D.M., and Lavoie, P., *IUTAM Bookseries*, Vol. 7, Springer, 2008.

[34] Li, C., Heinemann, P., and Reed, P., "Evolutionary Strategy (ES) to Optimize Electronic Nose Sensor Selection," *Computers in Agriculture and Natural Resources, 4th World Conference, Proceedings*, American Society of Agricultural and Biological Engineers, 2006.

[35] Igel, C., Erlhagen, W., and Jancke, D., "Optimization of Neural Field Models," *Neurocomputing*, Vol. 36, 2001, pp. 225-233.

[36] Büche, D., Guidati, G., and Stoll, P., "Automated Design Optimization of Compressor Blades for Stationary, Large-Scale Turbomachinery," *Proceedings of the ASME/IGTI Turbo Expo 2003*, 2003.

[37] Nagata, Y., "The Lens Design Using the CMA-ES Algorithm," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, Edited by Deb, K., Springer, 2004.

[38] Villasana, M., and Ochoa, G., "Heuristic Design of Cancer Chemotherapies," *IEEE Transactions on Evolutionary Computation*, Vol. 8, Issue 6, 2004, pp. 513-521.

[39] Hohmann, W., *Die Erreichbarkeit der Himmelsköper*, Oldenbourg, Munich, 1925; also available as NASA Technical Translation F-44, 1960.

[40] Vallado, D.V. with McClain, W.D., *Fundamentals of Astrodynamics and Applications*, 3rd ed., Microcosm Press and Springer, Hawthorne, CA, 2007, pp. 330-332.

[41] Escobal, P.R., *Methods of Astrodynamics*, Reprint ed., Krieger Publishing, Malabar, FL, p. 66, 1979.

[42] Lawden, D.F., *Optimal Trajectories for Space Navigation*, Buttersworth, London, 1963, Chapter 6.

[43] Barrar, R.B., "An Analytic Proof that the Hohmann-Type Transfer is the True Minimum Two-Impulse Transfer," *Acta Astronautica*, Vol. 9, No. 1, 1963, pp. 1-11.

[44] Hazelrigg, G.A., Jr., “The Use of Green’s Theorem to Find Globally Optimal Solutions to a Class of Impulsive Transfers,” American Astronomical Society, AAS Paper 68-092, Sept. 1968.

[45] Marec, J.P., *Optimal Space Trajectories*, Elsevier, Amsterdam, 1979, Chapter 2, pp. 21-27.

[46] Battin, R.H., *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, AIAA, New York, 1987, pp. 529-530.

[47] Palmore, J.I., “An Elementary Proof of the Optimality of Hohmann Transfers,” *Journal of Guidance, Control and Dynamics*, Vol. 7, No. 5, 1984, pp. 629-630.

[48] Prussing, J.E., “Simple Proof of the Global Optimality of the Hohmann Transfer,” *Journal of Guidance, Control and Dynamics*, Vol. 15, No. 4, 1992, pp. 1037-1038.